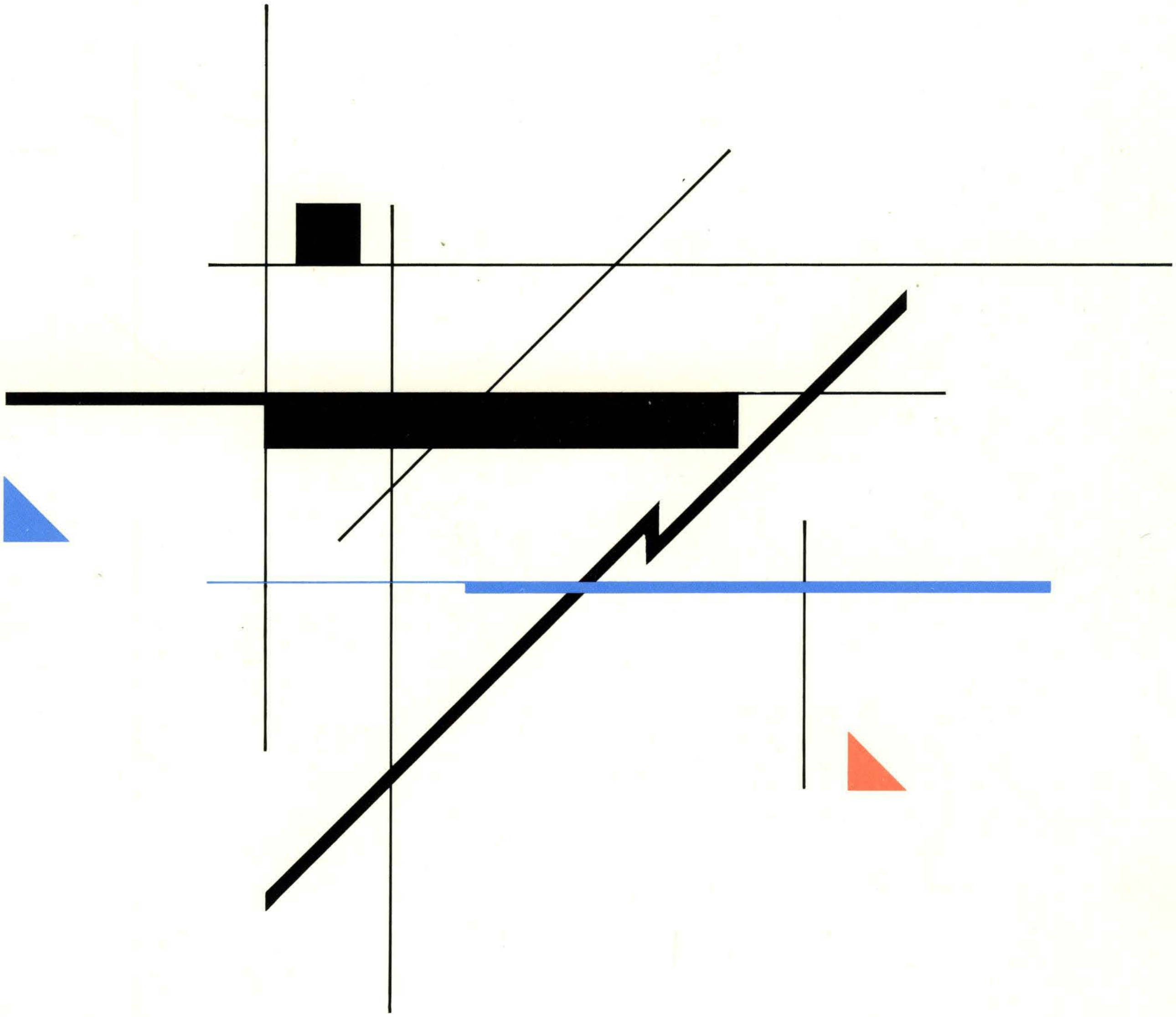
IBM

Systems Network Architecture Distribution Services

SC30-3098-3

Reference



Systems Network Architecture Distribution Services

SC30-3098-3

Reference

Fourth Edition (June 1989)

This is the fourth edition, SC30-3098-3; it applies until otherwise indicated in a new edition. Consult Part 3 of the latest edition of *IBM System/370, 30xx, and 4300 Processors — Bibliography*, GC20-0001, for current information on this communication architecture.

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

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for your comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation
Department E74
P.O. Box 12195
Research Triangle Park, North Carolina 27709, U.S.A.

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you or restricting your use of it.

Note to US Government users - Documentation related to Restricted Rights - Use, duplication, and disclosure is subject to restrictions set forth in GSA ADP Schedule contract with IBM Corp.

Special Notices

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to use these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, New York 10577.

IBM is a registered trademark of the International Business Machines Corporation.

Preface

This manual describes Systems Network Architecture/Distribution Services (SNA/DS) at the implementation level. This manual does not describe any specific machines or programs that may implement SNA, nor does it describe any implementation-specific subsets or deviations from the architectural description that may appear within any IBM SNA product. These matters, as well as information on SNA product installation and system definition, are described in the appropriate publications for the particular IBM SNA machines or programs to be used.

Prerequisite Publications

- *SNA Concepts and Products*, GC30-3072
- *SNA Technical Overview*, GC30-3073
- *SNA Transaction Programmer's Reference Manual for LU Type 6.2*, GC30-3084

Related Publications

- *SNA Formats*, GA27-3136
- *SAA Common Programming Interface: Communications Reference*, SC26-4399
- *SNA LU 6.2 Reference: Peer Protocols*, SC31-6808
- *SNA Type 2.1 Node Reference*, SC30-3422

Contents

Figures

Tables

Chapter 1. Concepts and Facilities

Introduction

SNA/Distribution Services (SNA/DS, or simply DS) provides a general-purpose, connectionless communications service to applications that use it. A *connectionless* service is one in which communication is performed without the establishment of a direct connection between (or among) the communicating parties. Such a service is also commonly known as a *messaging* service. In contrast, a *connection-oriented* service is one that does provide a direct connection between the communicating parties. DS is connectionless at the transaction services layer of SNA; from the DS perspective, an SNA session (or more precisely, an LU 6.2 conversation) between two application programs is an example of a direct connection between those programs.

DS allows application programs to communicate without requiring that the origin and destination of the communications both be active simultaneously. The architecture allows the nodes at which the origin and destination application programs reside (not those application programs themselves) to communicate via direct sessions. Alternatively, those nodes may communicate via intermediate nodes that provide a store-and-forward function. Traffic is queued, if necessary, before being sent from one node to the next.

The connectionless nature of the service does not necessarily imply long delays in completing the processing of requests. It does imply that the communicating application programs do not interact and therefore that responsibility for the unit of work cannot be shared but must be shifted. The originating application program transfers responsibility for a request to the distribution service, which subsequently transfers the request to the destination application program. Once the distribution service has accepted the request, it is independently responsible for carrying it out, perhaps within milliseconds, perhaps not for hours.

The distribution service (Figure 1 on page 2) consists of a network of nodes known as *distribution service units* (DSUs). This network may be thought of initially as occupying a geographic area of no particular shape. The entities that use the distribution service are outside the boundary of that area. The service accepts requests from, and makes deliveries to, those entities across that boundary.

The unit of work performed by the service is termed a *distribution*. A distribution begins at one DSU and may spread out to many. The work performed on a distribution includes the acceptance of the request at the origin, the generation and movement of copies of the distributed material across the network, and the delivery of those copies to the specified destinations. Various levels of service may be requested for distributions: for example, higher or lower priorities.

SNA/DS utilizes and complements the services provided by the lower layers of SNA. Distribution service units communicate with one another via LU 6.2 basic conversations. Users of DS perceive the distribution service as just part of the overall SNA services.

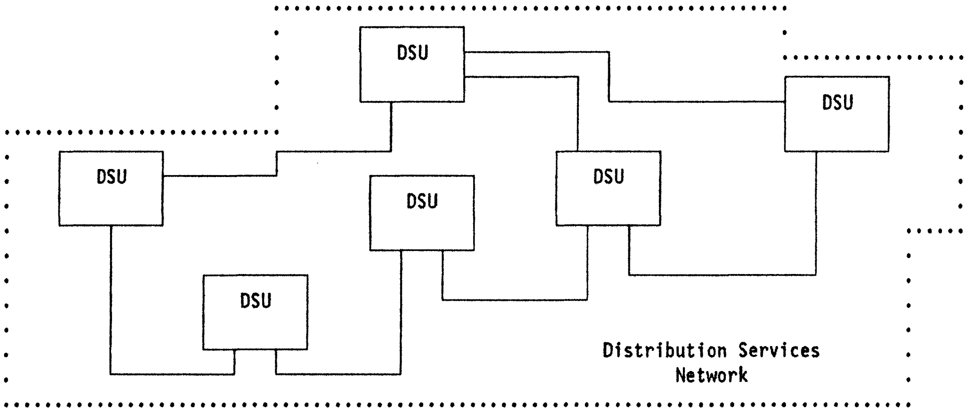


Figure 1. SNA/Distribution Services Network

The Interface to the Distribution Service

Agents

DS performs services in response to requests issued by application transaction programs. These application transaction programs are known as *agents*. Agents exist outside the boundary of the DS network (Figure 2 on page 4), and provide users and/or service functions with access to DS services. Agents may send distributions, receive distributions, or issue operations commands on behalf of their users. There is a many-to-many relationship between users and agents; that is, an agent is typically capable of acting on behalf of a variety of users, and users typically make use of a variety of agents.

Agent Requests

The originating agent requests that DS perform a distribution. The request specifies the data object (or objects) that is to be distributed, and specifies the name of the agent that is to be invoked at the destination(s) to process the distribution.

The request also specifies one or more destinations, which may be either users or DSUs or a mixture of both. (Similarly, a user (or a DSU) could receive distributions originated on behalf of other users or on behalf of DSUs.) An originator would specify a user as a destination to send information to that particular user. The originator need not know the locations of users; DS determines the location of each destination user specified. The originator would specify a DSU as a destination in order to send information to a particular location in the network, perhaps to a service function (not a user) at that location.

If the originator supplies a user name as a destination, the destination is referred to as a *user destination*. If the originator supplies a DSU name as a destination, the destination is referred to as a *node destination*.

Small objects to be distributed may be imbedded directly in the originator's request. Larger objects are typically not passed directly to DS by the agent, but are referred to by name or location so that DS can access them when needed.

Several other parameters are included in the request, some required, some optional. This chapter introduces certain key parameters. For a complete description, refer to Appendix F.

The Agent Protocol Boundary

The interfaces across which DS interacts with non-DS entities are called *protocol boundaries* (PBs). The particular interface across which DS interacts with agents and operators is called the *agent protocol boundary*. Other protocol boundaries defined by DS are the *server protocol boundary* and the *queue protocol boundary*. In addition, DS communicates with LU 6.2 via the LU 6.2 basic conversation protocol boundary.

A protocol boundary defines the functions provided by and expected by the components on either side of that boundary. It does not necessarily define the precise syntax of requests issued across it. For more information on the protocol boundaries defined by DS, see Appendix F. For more information on the LU 6.2 basic conversation protocol boundary, see the *Transaction Programmer's Reference Manual for LU Type 6.2*.

The information contained in requests made by agents across the agent protocol boundary is formally defined in DS by several protocol boundary verbs. The two basic verbs are:

1. Send_Distribution. This verb is issued by an agent to request that a distribution be performed. Its parameters include the destination users or DSUs and the name of the agent to be invoked at the destination to receive the distribution.
2. Receive_Distribution. This verb is issued by an agent that is activated at the destination to accept delivery of a distribution.

In addition, several other verbs provide variations of the basic sending and receiving capabilities. Refer to "Agent Protocol Boundary Verbs" on page 55 and to Appendix F for further information about agent protocol boundary verbs.

Agent Names

Agents have names. At any given location in the network, an agent name uniquely identifies a specific application program. (There may, however, be multiple instances of a particular agent at a particular location.) Typically, a given agent name is known at multiple locations, and there are many instances of that particular agent throughout the network. Agents may be either user-written or architecturally defined.

A requesting agent must specify the name of the destination agent that will be activated to receive the distribution on behalf of the destination users or DSUs.

There is only one destination agent per request, no matter how many destinations are specified.

The Transfer of Responsibility

When the originating agent requests that DS perform a distribution, the agent transfers responsibility for the distribution to DS. Once the distribution service has accepted responsibility for a request, it performs the distribution independently; the originating agent is no longer involved.

While DS has responsibility for the distribution, exceptions may occur. If the originating agent has so specified, exception reports will be generated and sent to a specified user or DSU. When the report arrives, an agent will be invoked to handle it; this agent may be a different instance of the originating agent or a completely different agent. Reporting is thus performed asynchronously. This contrasts with synchronous exchanges that occur between transaction programs that communicate entirely within one conversation.

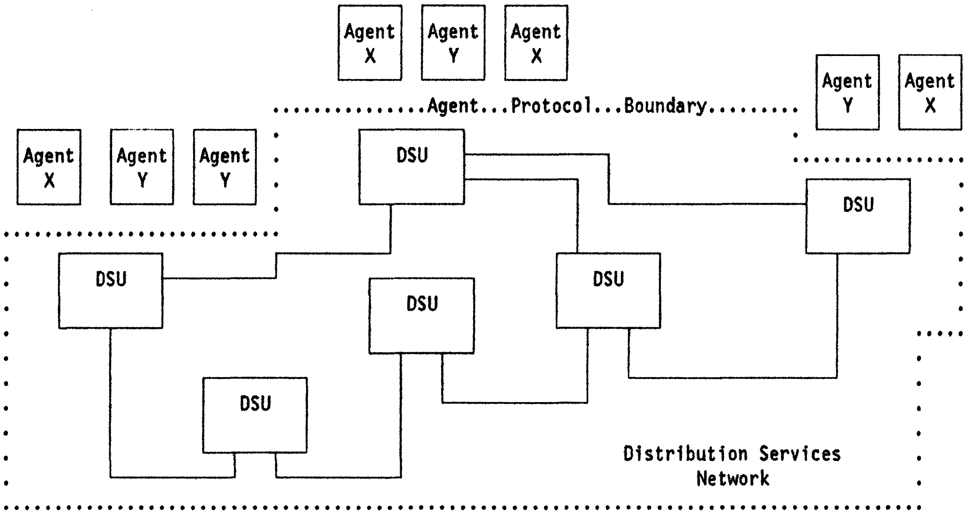


Figure 2. A DS Network Showing Agents

Distribution Service Users

Users of DS are defined as addressable entities on whose behalf agents can originate or accept delivery of distributions. Users may include individuals, departments, application programs, and data bases. Users of DS may include individuals or application programs with responsibilities for system and network operations or for installation and maintenance of network definition information.

User Roles

Users are referred to as *originating users* or *originators* when they request (via their agents) that a distribution be performed, and as *destination users* when distributions are delivered to them. Most users are capable of either role; however, it is possible for a user such as a data base to serve only as a destination.

User Names

DS users have names that are unique within the DS network. A user name consists of two parts, each of which can be up to eight bytes long. The two parts are known as the *distribution_group_name* (DGN) and the *distribution_element_name* (DEN). Each element name is unique within its group, and each group name is unique within the network.

Users, or their agents, must know the names of all the other distribution service users to whom they wish to send distributions. To facilitate this, user names could be publicized throughout an organization, exchanged over the phone, and included on letterheads.

An organization assigns group names based on whatever structure is most natural for it. Divisions or departments might be convenient group names; last names might be convenient element names. For example, Harry Chase in the Operations department could be OPS.CHASE, and Ellen Piaf in the Marketing department could be MKT.PIAF.

Users, however, are not necessarily people. For example, there might be a statistics data base in manufacturing to which various plants routinely ship data. It could have MFG.DBASE as its user name. Other user names might identify departments (BIGBANK.LOANS), applications (ACCTG.PAYROLL), or titles of positions (DEPTX.MGR).

Users in the same group (i.e., with the same DGN) need not be located near one another in the network. Furthermore, members of several groups might reside at the same location in the network. For example, at Paris, there might be users in the loans (LOANS.PIERRE), payroll (PAYROLL.PORTER), and personnel (PER.PEDRO) departments. Other users in these same departments could be located throughout the network; for example, PAYROLL.CHARLES might be located in Chicago, and PAYROLL.NORTON might be located in New York.

DS user names are *location-independent*. When installations select user names with no suggestion of location, users can be moved from one computer system to another without needing to change their names. Since DS does not tie user names to particular locations, users with location-oriented names could also be moved without changing their names, but it would be confusing to continue to refer to someone as CHICAGO.SMITH after he had moved to Atlanta.

User's View of the Distribution Service

The user's view of the distribution service is shown in Figure 3. The user names are all unique, and are scattered around the periphery of the network with no regard to location. The agent names are not unique; numerous agent instances exist for each agent name.

When users request distributions with no server objects (a concept discussed later), user names and agent names are the only names of which users need be aware. When server objects are involved, the users (or their agents) must be aware of certain other additional names. These are discussed in "Servers and Objects" on page 40.

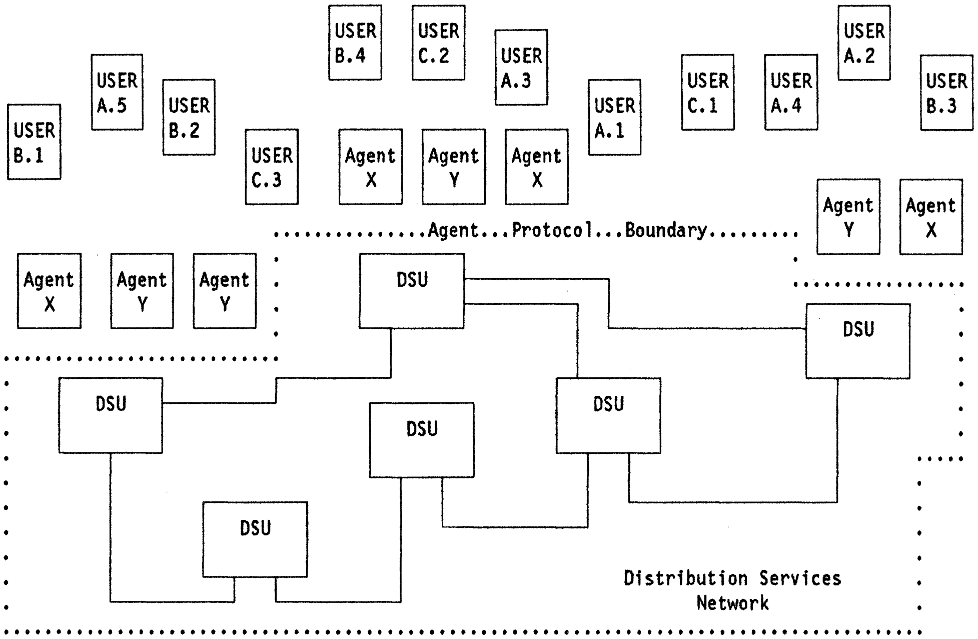


Figure 3. User's View of the Distribution Service

Distributions

A distribution is the unit of work performed by DS. The distribution starts as a request made by the originating agent, and continues through when the distributed material is delivered. If the agent requests notification on the status of the distribution, DS generates *distribution reports* to provide such notification. Distribution reports are considered part of the same unit of work (i.e., the distribution) as the agent's request.

Agents may make distribution requests on behalf of either users or the DSUs at which the agents reside, and may specify destinations that are either users or DSUs. When multiple destinations are specified, DS provides, or arranges access to, a copy of the distributed material in a machine-readable form (on disk storage, for example) for each of the named destinations.

Distribution Copies

From the perspective of the originator, a distribution includes the delivery of a copy to every destination on the destination list. From the perspective of a particular destination, the distribution consists of one delivery. Different points in the DS network may deal with different subsets of the original list of destinations. In summary, therefore, a distribution refers to work being done upon one, some, or all of its copies, depending on the perspective.

Types of Information in a Distribution

A distribution contains essentially two types of information: the "application" information that the agent has submitted to DS for distribution, and the DS control information that flows along with and encloses the application information. The distinction between these two is analogous to the distinction between the pages of a letter and the envelope that encloses them. The DS control information is analogous to the name, address, and handling instructions written on the envelope. When the distribution flows across the network, the application information is clearly separated from the control information; that is, it is contained inside the "envelope."

Distribution Transport Message Units

DS uses Distribution Transport Message Units (DTMUs) to transport the originator's information to the destinations named in the distribution request. Whereas a distribution is thought of as flowing through a network, possibly through several DSUs, a DTMU is thought of as existing only between adjacent DSUs. That is, the DTMU is "born" when it is encoded at a particular DSU; it is then transmitted to an adjacent DSU, and "dies" when it is decoded by that DSU and converted to an internal format (such as a data structure).

The structure of a DTMU is shown in Figure 4 on page 8. The DTMU is introduced by a prefix and concluded by a suffix. The DS control information for the distribution is contained in the command; the names of the users or DSUs for which this particular copy of the distribution is destined are encoded in the destination list.

The information submitted for distribution by the originator is contained in the agent and/or server objects. The *agent object* is intended for small amounts of data that can be stored by DS and passed directly to the destination agent. Larger amounts of data, or data that requires a particular kind of handling (encryption, for example, or specialized parsing) usually flow in the *server object*. Agent and server objects will be discussed further under the section "Servers and Objects" on page 40.

A detailed description of the encoding for DS message units is given in Appendix G.

The layers of SNA below DS may divide DS message units (MUs) into smaller pieces or assemble several small DS MUs into a single large piece, but DS is unaware of such manipulations. No direct relationship exists between segmenting performed by DS and the techniques used by lower layers of SNA.

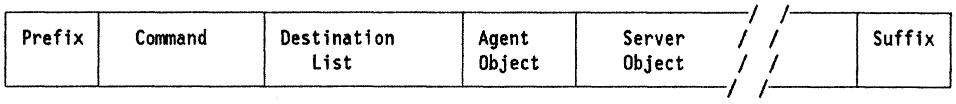


Figure 4. Structure of the Distribution Transport Message Unit

Distribution Report Message Units

As part of the distribution request, the originator may ask that DS provide feedback on the status of the distribution. For example, the originator might wish to be informed if DS is unable to deliver the distribution. DS sends such feedback information in distribution reports, which flow through the network in Distribution Report Message Units (DRMUs). The structure of DRMUs differs from that of DTMUs. DS reporting and DRMUs are discussed in the section "Distribution Reporting" on page 66.

The term *distribution message unit* (DMU) is sometimes used to refer to either a DTMU or a DRMU.

The Distribution Identification

Each distribution in the DS network is uniquely identified by a combination of fields carried in the DTMU. These fields are known collectively as the *distribution identification* (*dist_ID*). The *dist_ID* is composed of the name of the originating agent, the name of the DSU at which the distribution was originated, the name of the user, if any, on whose behalf the distribution request was made, the date of the distribution, and a sequence number.

Sequence number counters are maintained for each user-agent combination; that is, a different sequence number counter is kept for each combination of local user and local agent name. An additional counter is kept for each local agent that may originate distributions on behalf of the DSU itself (with no origin user involved).

DS Format Sets

Previous implementations of DS have used a different set of encoding rules for distribution message units. The earlier encoding for DMUs is referred to as DS Format Set 1; the current set of encodings is referred to as DS Format Set 2. Both sets of encodings are documented in Appendix G. The rules that allow these two format sets to coexist are documented in Appendix D.

Distribution Service Unit (DSU)

A distribution service unit (DSU) is the collection of transaction programs and data structures that provide the distribution service at any given location in a DS network. These transaction programs are distinct from application transaction programs such as those that issue requests to DS. The DS transaction programs are examples of SNA *service transaction programs.*

DSU Roles

DSUs have roles that vary with each distribution they service. The DSU at which a distribution request originates is the *origin DSU* for that distribution. The same DSU would have the role of *destination DSU* for a distribution sent to users located at that DSU. Alternatively, the DSU may perform a purely intermediate role. In this case, distributions are received and stored at the *intermediate DSU*, then forwarded to other DSUs.

A DSU may perform multiple roles for a distribution. For example, a user at a DSU might send a distribution to another user at the same DSU. The DSU, in that case, would be both the origin and the destination of that distribution.

DSU Names

DSU names consist of two parts, each of which can be up to eight bytes long. The two parts are known as the routing group name (RGN) and the routing element name (REN). Each element name is unique within its group, and each group name is unique within the DS network. Typically, the names will be assigned to be meaningful to systems programmers and operations people, not to the user population. A DSU includes its unique DSU name in all distributions it originates.

The DSU/User Relationship

Every DS user attaches to the DS network at a DSU. Typically, a DSU has several users, although it is possible for DSUs to have no users.

The DSU is not aware of anything above the protocol boundary it has with application transaction programs other than the names of its users, agents, and certain control information used to deliver distributions to them. The users themselves may be physically remote from the processor containing their DSU, but as far as DS is concerned they are located at that DSU.

Environment of the DSU

Figure 5 on page 10 illustrates the various entities with which a DSU interacts. Pictured above the DSU are those entities that issue commands to it. The agent is an application program that requests DS services. The *operator* issues commands to perform system or network maintenance functions.

DS interacts with *servers* in order to access large data objects for distribution. The server provides storage of objects that the DSU receives in distributions; it retrieves objects to be sent in outbound distributions.

DSUs communicate with one another via LU 6.2 basic conversations; they rely on the local operating system for facilities such as queue handling.

Subsequent sections of this chapter will explore the DSU's relationship to each of these components. A detailed description of the various protocol boundaries is given in Appendix F. DS's usage of LU 6.2 basic conversation verbs is documented in Chapter 2.

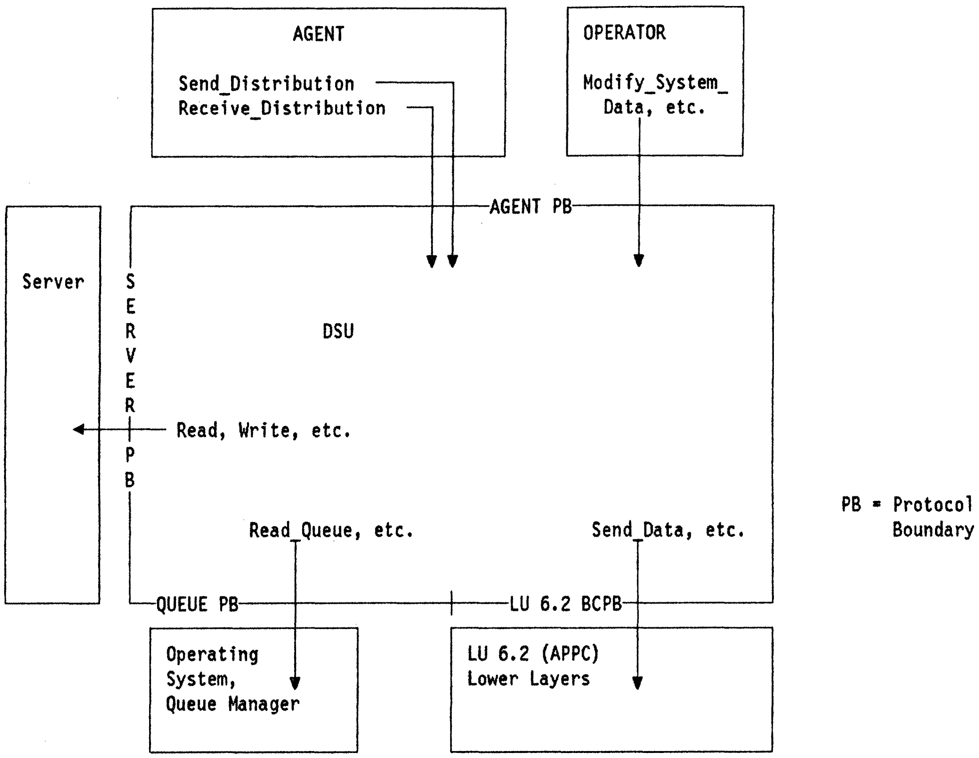


Figure 5. Environment of the DSU

DSU Directories and Routing Tables

A DSU's Directory

The DSU directory includes certain information for each DS user. (Directories shared with other functions might contain other information of which DS would be unaware). The directory entries for local users contain information used for DS delivery to those users (for example, a local queue name). Entries for users at other DSUs usually contain the name of the DSU at which they are located. Exceptions to this are discussed in "Redirection" on page 35.

The number of users in the network can be much larger than can be conveniently contained in the directory of a particular DSU. See "Default Directing" on page 36 to learn how distributions can be directed in cases where the origin DSU's directory does not contain entries for the destination users.

Users need not be aware of the names of DSUs at which other users reside. Users specify only the user names to which a particular distribution is to be sent; DS uses the directory to map those user names to DSU names.

When organizations set up their user names appropriately (that is, with no location or DSU implications), users can be moved from one DSU to another without having to change their names. For example, in large networks, groups of users are often shifted from one computer system to another because of office space rearrangements, or in order to balance loads on computing equipment. The impact of such changes is confined to the directories. The DSU name for each affected user is changed, but the user name itself is not changed. This means that user names can be completely insulated from the DSU name change and can be published or otherwise disseminated throughout an organization without concern for obsolescence due to system changes.

User names themselves may change from time to time. However, if they are appropriately assigned, those changes would be the ones of which other users should normally be aware. For example, if a user's department were part of a user name and the user changed departments, then the user's name would have to be changed. Users throughout the network would have to be notified, but it is likely that those users would need to know about the job change anyway (and change their distribution requests accordingly).

Implementations may allow directory entries to be subdivided by agent name. That is, instead of one entry per user, the directory might have several entries per user, each of which identifies a different combination of user name and agent name. At a destination DSU, such entries might allow distributions for a particular user to be delivered to different local delivery queues, based on the destination agent name.

In addition to the entries for users, directories may contain an entry for the omitted user name. During the directing process, any destination for which no user name is specified (i.e., a node destination) would match such an entry. Like user name entries, an entry for the omitted user name may be subdivided by agent name.

User Aliases: To DS, user names represent only entries in the directory. Installations can assign user names to entities in any manner they wish. Aliasing is an interesting illustration of how such assignments could be made.

An individual can be given more than one user name. DS cannot tell when this has been done. It "sees" different users because each name has its own independent entry in the directory. When the two entries have the same local delivery information, distributions for either name are delivered to the same individual. For example, DEPT72.MGR and EMPNO.X12345 could both be defined in the directories so that local delivery was made to Harry Jones. Other users interested in the work of department 72 would probably use DEPT72.MGR. Users in Personnel or Payroll would probably use EMPNO.X12345.

If Harry Jones were to be transferred to another job at another location, the EMPNO.X12345 directory entries would be updated with his new DSU name. The DEPT72.MGR name, on the other hand, would probably be reassigned to the person who replaced him.

User Names vs. Nicknames: DS user names are not to be confused with nicknames. For reasons of convenience, or perhaps some system limitation, it is often desirable to refer to users by other, usually shorter, names. For example, an originating user might not wish, or might not be able, to logon to his system with a 16-byte name. Also, he might not wish to enter 16-byte names for the destination users, particularly those to whom he sends things frequently. In such cases, nickname files can be used. Such files are not part of DS. The nicknames would not flow in DS MUs, except perhaps as part of the user's data. The nicknames would be unique only within a DSU or perhaps only for one user.

Nickname files are not defined by the DS architecture, but they would need to contain the full network-unique DS user name for each nickname. The origin agent might access the nickname file to obtain the DS user name that would be included in the distribution request. DS, however, would be unaware that such files existed. The complete DS user name flows over the network and is used in exception reporting. The users are aware, not only of local nicknames, but also of complete user names, both their own and those of others to whom they wish to send distributions.

The overall responsibility for the creation and maintenance of the directories would usually be the system administrator's. In a large organization, this could be a major task. The two-part user name and default directing (see "Default Directing" on page 36) can be used to allocate this responsibility by group name. For example, if the DGNs were departments, each department could be made responsible for its own set of DENs.

A DSU's Routing Table

In the simplest case, the routing table consists of one entry for each destination DSU. Each entry identifies the connection to be used to send distributions to that particular destination. The routing table and the directory serve distinctly different purposes. The directory indicates where a user is located; the routing table indicates how to get there--that is, which direction to go.

Refer to "Distribution Service Parameters" on page 23 and "Default Routing" on page 38 for a more complete description of the routing table entries.

The creation and maintenance of routing tables is the responsibility of the system administrator. Like directory maintenance, this is a significant task. Although a typical network could have 10 to 100 times more users than DSUs, the number of routing tables in which each DSU name appeared would be much greater than the number of directories in which the typical user name appeared. Default routing allows the number of entries in the routing tables to be considerably reduced. With default routing, a distribution may be routed to a larger DSU which would have a more complete routing table. The use of default routing is described in "Default Routing" on page 38.

Simple Networks

A DS network is a collection of two or more DSUs and the connections between them. A DS *connection* is the set of actual or potential LU 6.2 conversations, using a particular LU 6.2 mode name, between two DSUs. DS connections share the underlying path control network with sessions belonging to other applications. A DS connection may consist of one or more than one LU 6.2 conversation.

Fully-Connected DS Networks

A fully-connected DS network is one in which every DSU has a connection to every other DSU. Networks can be set up this way if the underlying layers of the SNA network provide total interconnectability. We will use a very simple fully-connected network (Sample Network #1, Figure 6) to illustrate the concepts of simple directing and routing.

There are four users in sample network #1, located in three cities. Their names are listed by department under their cities. The boundary of the DS network is shown intersecting three boxes, one in each city. The boxes represent processors; the portion of each processor that is inside the DS boundary includes a DSU. Each DSU is labeled with its DSU name.

The small boxes inside each DSU represent a user directory and a routing table. The portion of the DS boundary that intersects each processor is the agent protocol boundary in that processor. The lines connecting the DSUs are DS connections, and are identified in Figure 6 by the pair of names of the two DSUs they connect.

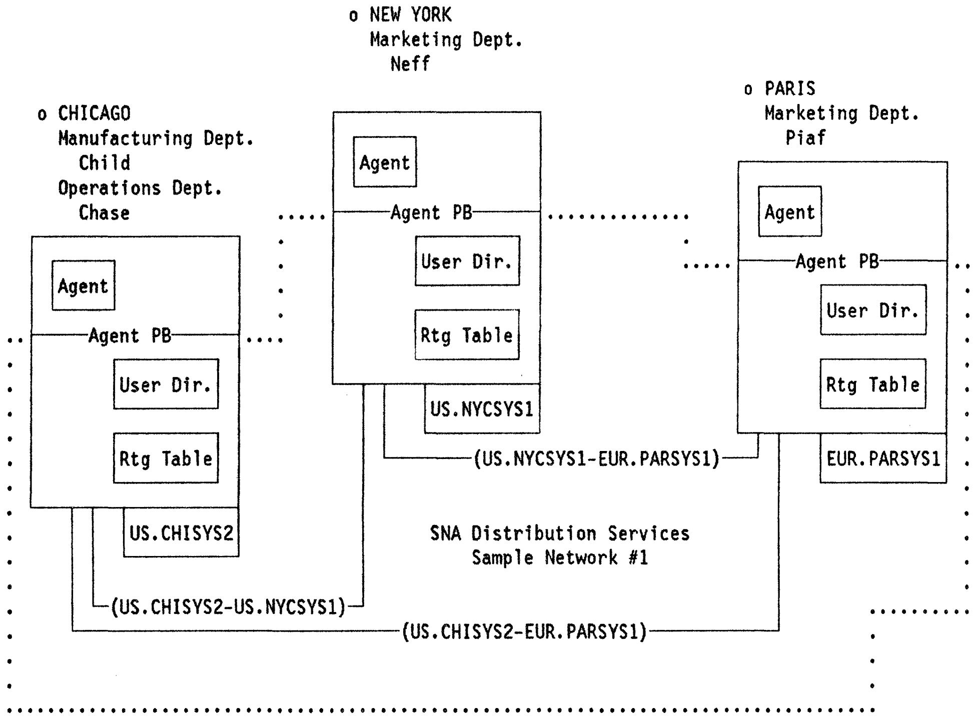


Figure 6. Sample Network #1

Simple Directing in Fully-Connected Networks

The *directing* function is the process of associating a destination location name with a destination user name. In the mail analogy, it would be the process of adding the address to the addressee's name on the envelope. In DS, it is the process of associating either a DSU name or local delivery information with every user name in the distribution.

The simplest kind of directing occurs when every DSU has a complete directory of all users. The following discussion presumes that. In practice, such a simple situation would probably never occur. The more sophisticated kinds of directing are described in "Redirection" on page 35 and "Default Directing" on page 36.

At the origin, the directory is used to obtain the corresponding destination DSU name for each destination user name in the distribution. Both the user names and the corresponding DSU names are then included in the control information that flows in the distribution. Directing is bypassed at the origin for node destinations.

At the destination, the directory is used to obtain the information needed to deliver the distribution. This information is used locally only; therefore, it is not defined by the DS architecture. Typically, it would consist of a queue identifier; different queues would be used to deliver distributions to different users. Directing is performed at the destination DSU for both user and node destinations.

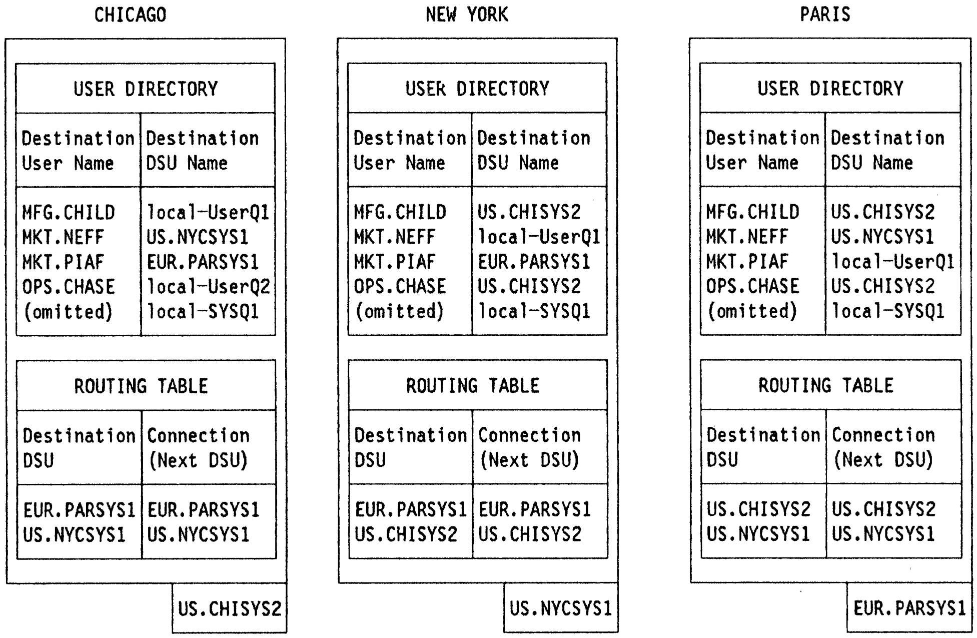
Figure 7 on page 15 illustrates the directories and routing tables for sample network #1. Recall that directories may contain an entry for the omitted user name; note the entry at US.NYCSYS1 for "omitted." The significance of this entry is that a distribution with an entry in the destination list specifying US.NYCSYS1 (no destination user name), when received at New York, would be delivered to the local queue SYSQ1.

Simple Routing in Fully-Connected DS Networks

Routing is the determination of the next route segment on which a distribution is to be sent, and the scheduling of or enqueuing for the sending activity.

The simplest kind of routing occurs when every DSU's routing table contains entries for all other DSUs. The following discussion presumes that to be the case. More sophisticated routing is described in "Default Routing" on page 38.

Each entry in the routing table identifies the DS connection over which distributions are to be sent in order to reach a particular destination DSU. In this illustration (Figure 7), with only one type of DS traffic, the connections can be uniquely identified by the pair of names of the DSUs they connect; however, in the routing table of one DSU only the name of the other DSU is required. This is contained in the column labeled "Next DSU." There is no particular DS definition of connection identifiers, so implementations may use different identifiers.



CHICAGO

USER DIRECTORY

Destination User Name	Destination DSU Name
MFG.CHILD	local-UserQ1
MKT.NEFF	US.NYCSYS1
MKT.PIAF	EUR.PARSYS1
OPS.CHASE	local-UserQ2
(omitted)	local-SYSQ1

ROUTING TABLE

Destination DSU	Connection (Next DSU)
EUR.PARSYS1	EUR.PARSYS1
US.NYCSYS1	US.NYCSYS1

US.CHISYS2

NEW YORK

USER DIRECTORY

Destination User Name	Destination DSU Name
MFG.CHILD	US.CHISYS2
MKT.NEFF	local-UserQ1
MKT.PIAF	EUR.PARSYS1
OPS.CHASE	US.CHISYS2
(omitted)	local-SYSQ1

ROUTING TABLE

Destination DSU	Connection (Next DSU)
EUR.PARSYS1	EUR.PARSYS1
US.CHISYS2	US.CHISYS2

US.NYCSYS1

PARIS

USER DIRECTORY

Destination User Name	Destination DSU Name
MFG.CHILD	US.CHISYS2
MKT.NEFF	US.NYCSYS1
MKT.PIAF	local-UserQ1
OPS.CHASE	US.CHISYS2
(omitted)	local-SYSQ1

ROUTING TABLE

Destination DSU	Connection (Next DSU)
US.CHISYS2	US.CHISYS2
US.NYCSYS1	US.NYCSYS1

EUR.PARSYS1

Figure 7. Directories and Routing Tables for Sample Network #1

MU Flows for Typical Distributions

A Single-Destination (Non-Local) Distribution: In Paris, Piaf in Marketing wishes to send a message to Chase in Operations. Her agent requests a distribution, identifying the originating user as MKT.PIAF, the destination user as OPS.CHASE, and the destination agent as agent X. The DSU at Paris consults its directory and determines that OPS.CHASE is at US.CHISYS2. The destination US.CHISYS2 is then used to determine from the routing table the connection for which the distribution should be enqueued. In Figure 8 on page 16, the box labeled DTMU-A represents the DTMU that flows. Only the control information pertinent to this discussion is depicted in DTMU-A.

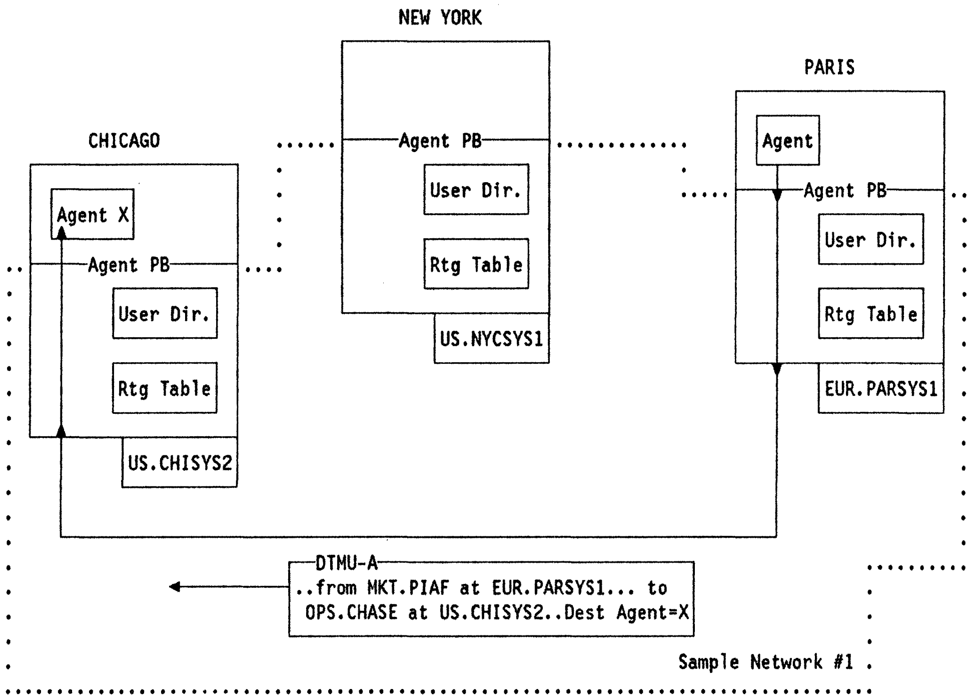


Figure 8. Single-Destination Distribution

A Multi-Destination Distribution Fanned Out at the Origin: In Paris, Piaf requests that a message be sent to Chase in operations and Neff in Marketing. As in the single-destination example, the DSU at Paris determines the destination DSU names and uses them to determine the routing. In this case, however, there are two destination DSUs. The DSU at Paris therefore sends two copies of the distribution as shown in Figure 9 on page 17. Notice that the control information in the DTMUs depends on the destination. The process of creating additional copies of a distribution is known as "fan-out."

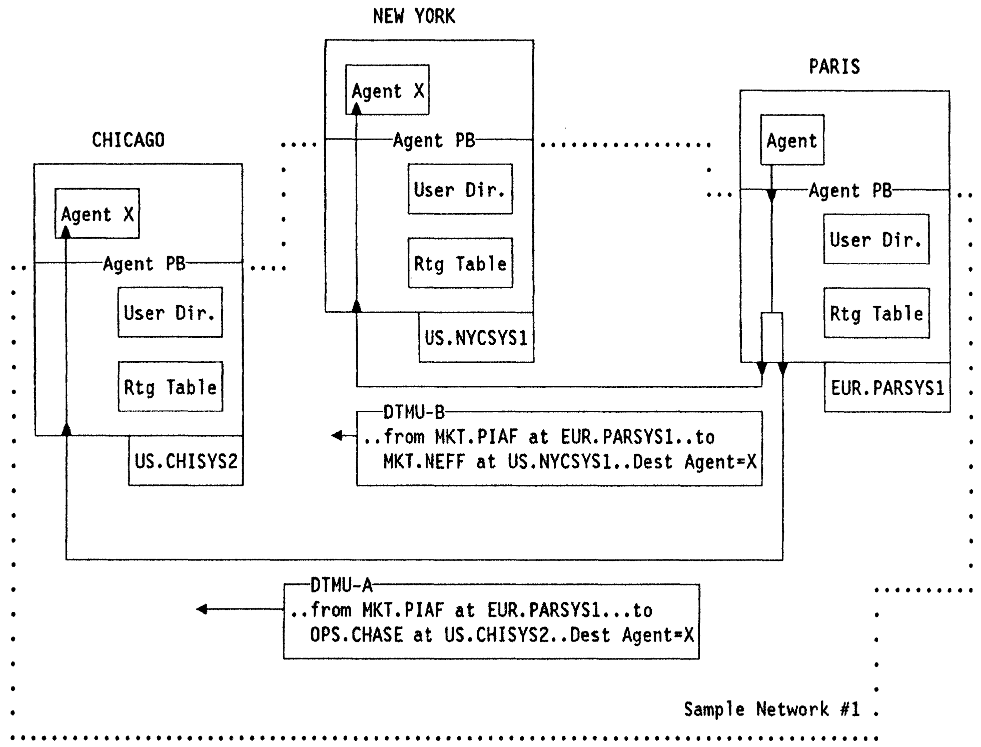


Figure 9. Multiple-Destination Distribution with Origin Fan-out

A Distribution Fanned Out at the Destination: In Paris, Piaf requests that a message be sent to the DSU at New York, with copies to Child in manufacturing and Chase in operations. The Paris DSU consults the directory for the two user destinations (MFG.CHILD, OPS.CHASE) and discovers that MFG.CHILD is at the same DSU as OPS.CHASE. It therefore sends only one DTMU to US.CHISYS2 and includes both user names in it. Refer to Figure 10 on page 18 and notice the contents of DTMU-A.

No directing is performed at the origin for node destinations, since the originator has already supplied the DSU name. Thus for the destination US.NYCSYS1, no directing is necessary. The routing table is consulted to determine the connection to use for US.NYCSYS1.

Since Child and Chase each have their own queues, the Chicago DSU creates an extra copy of the distribution. It places one copy in queue USERQ1 and the other in queue USERQ2. In some systems, users might be able to share the same copy of the distribution and there would be no need for the copying step. If Child and Chase happened to share the same queue, DS would make a single delivery containing both user names.

At New York, the distribution copy destined for US.NYCSYS1 is received. Directing is invoked; since there is no user name for this destination, the entry for "omitted" is matched. The distribution is placed in queue SYSQ1, the destination agent X is started, and the distribution is passed to it.

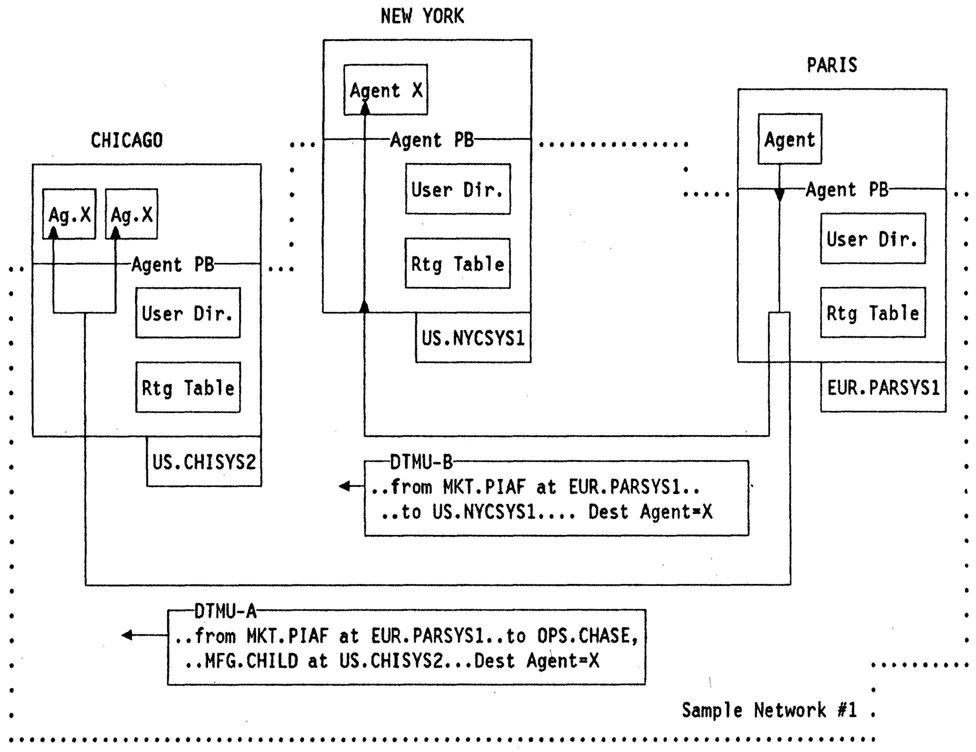


Figure 10. Multiple-Destination Distribution with Destination Fan-out

DS Networks with Intermediate DSUs

This type of DS network differs from the simple type discussed above in that the DSUs are not fully connected. In order for distributions to travel between certain DSUs, other DSUs must perform an intermediate role. The use of intermediate DSUs to forward distributions provides functional and performance advantages.

The sessions used by DS connections sometimes span multiple path control (PC) intermediate routing nodes (IRNs). In these cases, the PCIRNs are often geographically close to a node containing a DSU. For example, consider sample network #1 in Figure 6 on page 13. The processor in New York that contains the DSU is a System/370. Directly attached to that processor is a communication controller, through which the sessions used by the (EUR.PARSYS1-US.CHISYS2) connection pass. This is illustrated in Figure 11 on page 19. The components of the path control network are depicted by dotted lines. The presence of the PCIRNs is transparent to DS; the DSU sees only the direct session to its partner DSU.

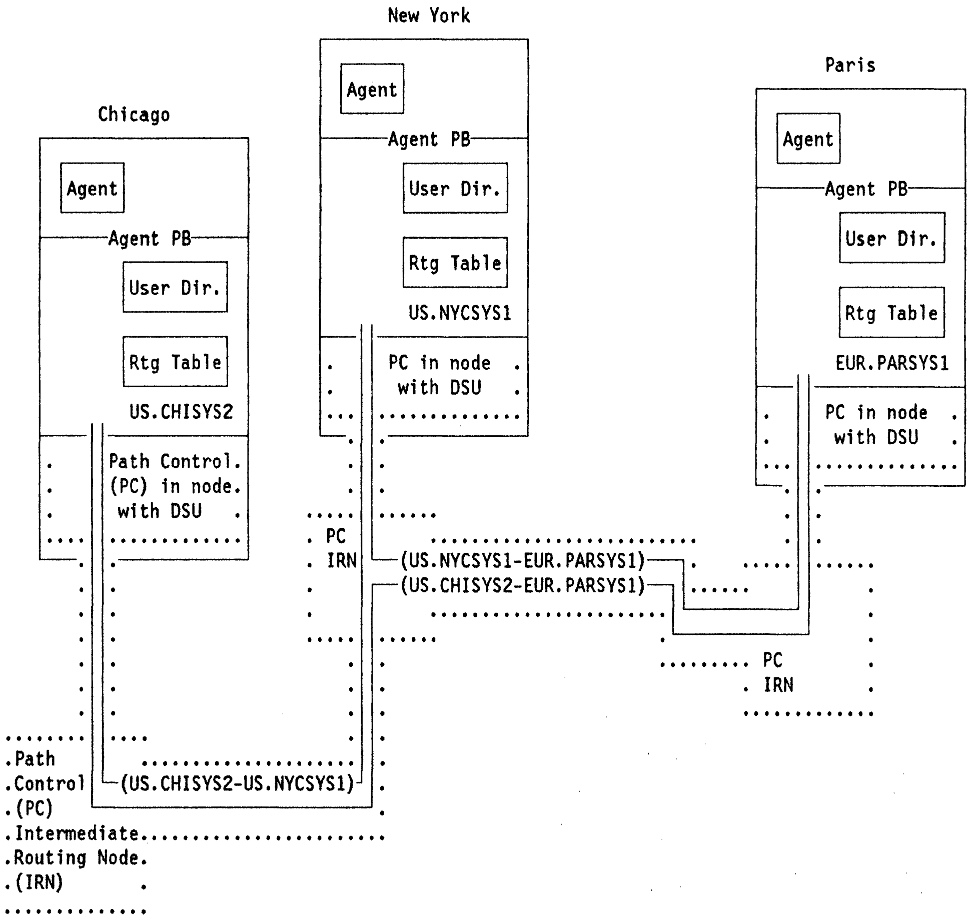


Figure 11. Sample Network #1 Showing Underlying Path Control Network

In the case of the distribution from Piaf at Paris to Neff at New York and Chase at Chicago, two copies of the one distribution travel across the same transatlantic link (the dotted connection between the PCIRNs in Figure 11). For large distributions, such duplication is rather wasteful.

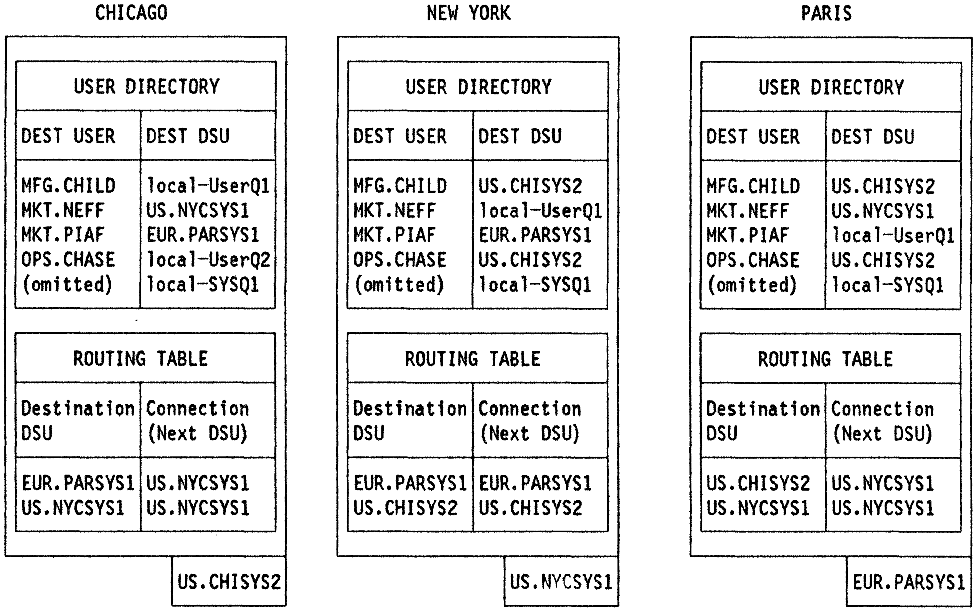
To avoid this inefficiency, large distributions destined for US.CHISYS2 are sent first to US.NYCSYS1, where they are received completely and then forwarded. The direct DS connection between EUR.PARSYS1 and US.CHISYS2 in sample network #1 is broken into two shorter ones. The two-connection version of the network is shown as sample network #2 in Figure 13 on page 21.

Simple Directing in Networks with Intermediate DSUs

Directing in this type of network at the origin and destination DSUs is the same as in fully-connected networks. In the simple case, directing is not invoked at intermediate DSUs. The routing function is all that is involved at a DSU performing a purely intermediate role. An exception to this is described in "Redirection" on page 35.

Simple Routing in Networks with Intermediate DSUs

In this type of DS network, the structure of the routing tables is the same as in a fully-connected one. That is, each entry identifies the connection over which distributions are to be sent in order to reach a particular destination DSU. The difference is that, for a route involving intermediate nodes, the connection identified in the entry does not connect the origin and destination DSUs. For example, refer to Figure 12. In Chicago, the entry for EUR.PARSYS1 identifies a connection with US.NYCSYS1; in Paris, the entry for US.CHISYS2 identifies a connection with US.NYCSYS1. The New York routing table remains the same as in sample network #1.



CHICAGO

USER DIRECTORY	
DEST USER	DEST DSU
MFG.CHILD MKT.NEFF MKT.PIAF OPS.CHASE (omitted)	local-UserQ1 US.NYCSYS1 EUR.PARSYS1 local-UserQ2 local-SYSQ1

ROUTING TABLE	
Destination DSU	Connection (Next DSU)
EUR.PARSYS1 US.NYCSYS1	US.NYCSYS1 US.NYCSYS1

US.CHISYS2

NEW YORK

USER DIRECTORY	
DEST USER	DEST DSU
MFG.CHILD MKT.NEFF MKT.PIAF OPS.CHASE (omitted)	US.CHISYS2 local-UserQ1 EUR.PARSYS1 US.CHISYS2 local-SYSQ1

ROUTING TABLE	
Destination DSU	Connection (Next DSU)
EUR.PARSYS1 US.CHISYS2	EUR.PARSYS1 US.CHISYS2

US.NYCSYS1

PARIS

USER DIRECTORY	
DEST USER	DEST DSU
MFG.CHILD MKT.NEFF MKT.PIAF OPS.CHASE (omitted)	US.CHISYS2 US.NYCSYS1 local-UserQ1 US.CHISYS2 local-SYSQ1

ROUTING TABLE	
Destination DSU	Connection (Next DSU)
US.CHISYS2 US.NYCSYS1	US.NYCSYS1 US.NYCSYS1

EUR.PARSYS1

Figure 12. Directories and Routing Tables for Sample Network #2

This slight difference in content reflects a significant difference in concept. In DS, a route is defined as the sequence of DSUs through which a distribution has traveled when it arrives at its destination DSU. The routing function is performed independently at each DSU. At any particular DSU, the routing function does not select a route in its entirety (except where there happens to be a direct connection). The routing function actually selects a route segment. The notion of route segment differs from the notion of connection in that segments of more than one route may use the same connection. In sample network #2, the connection EUR.PARSYS1-US.NYCSYS1 is used by segments of two routes, one from EUR.PARSYS1 to US.NYCSYS1 and the other from EUR.PARSYS1 to US.CHISYS2.

Because, in some implementations, operators could change routing tables as distributions move through the network, the route for any particular distribution cannot be reliably predicted at its origin DSU. Each distribution finds its own way across the network, one route segment after another. Two distributions from the same origin might find different routes to the same destination.

MU Flows for Typical Distributions

Single Destination Through an Intermediate DSU: Figure 13 shows the two-connection network (sample network #2) with a single-destination distribution flowing from Paris to Chicago. The DSU at Paris determines that distributions for Chicago should be sent on the connection to New York. When the DSU at New York receives the distribution, it examines the destination list and discovers that there is no local destination. It then uses its routing table to forward the distribution on the connection to Chicago.

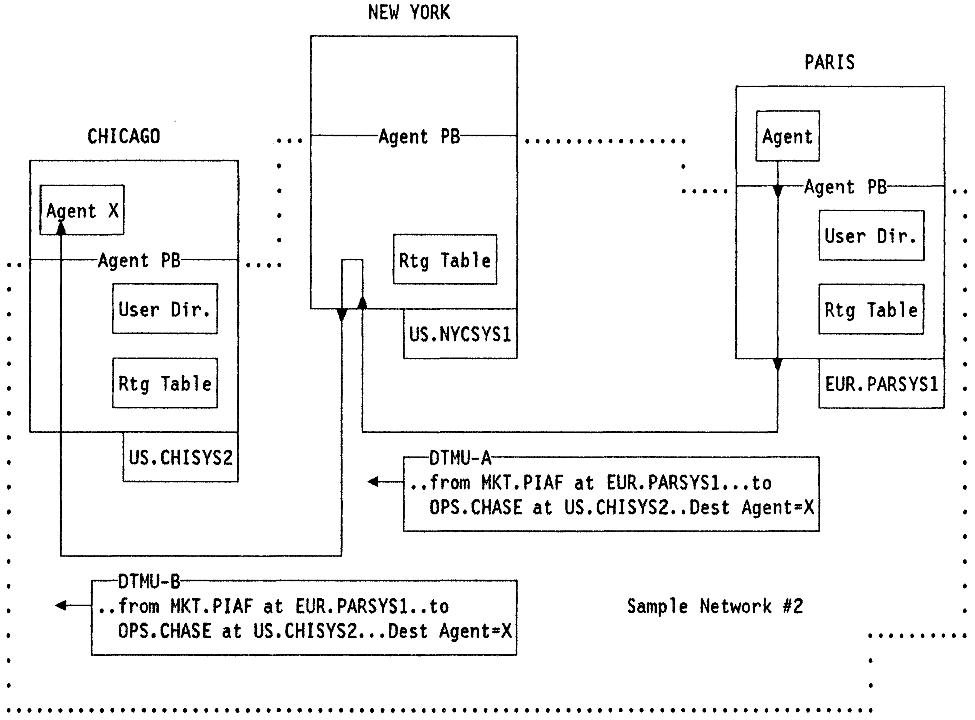


Figure 13. Single-Destination Distribution Through an Intermediate DSU

Distribution Fanned Out at an Intermediate DSU: In the two-connection network (sample network #2), the distribution from Piaf in Paris to the DSU in New York and Chase and Child in Chicago is not fanned out by the Paris DSU. When the DSU determines that distributions for US.CHISYS2 should be sent to US.NYCSYS1, it sends only one copy over the transatlantic link (see Figure 14 on page 22). In New York, the DSU delivers one copy of the message (for the node destination US.NYCSYS1) to the destination agent X, and forwards another copy to US.CHISYS2.

Notice how the control information in DTMU-B differs from that in DTMU-A. The US.NYCSYS1 destination information is stripped out when the distribution goes through the intermediate DSU.

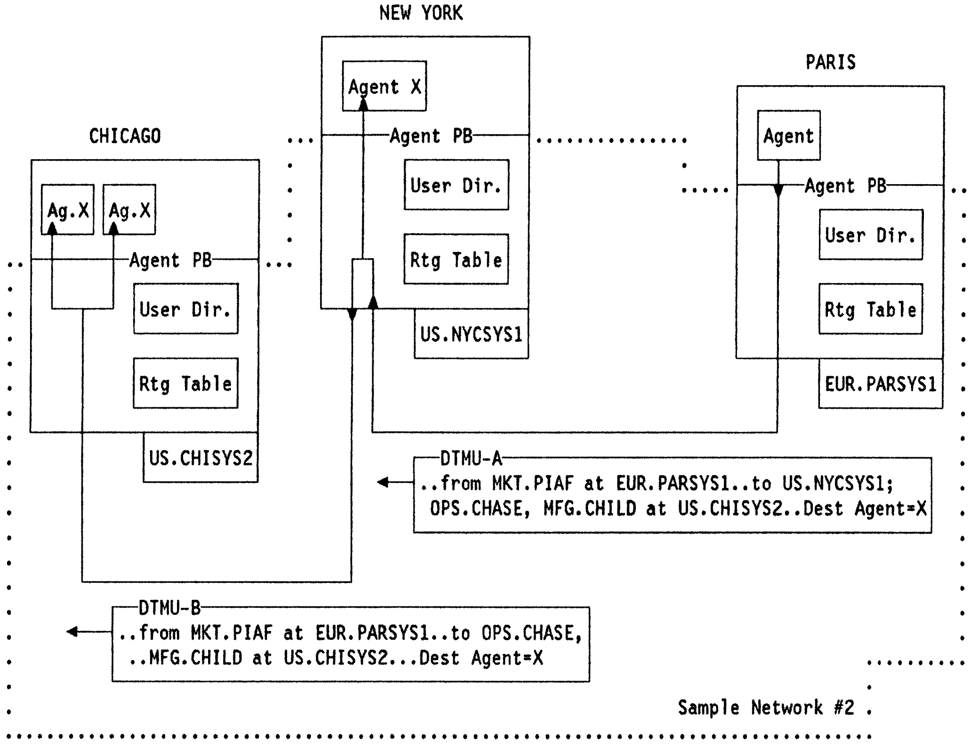

NEW YORK
Agent X
Agent PB
User Dir.
Rtg Table
US.NYCSYS1
PARIS
Agent
Agent PB
User Dir.
Rtg Table
EUR.PARSYS1
CHICAGO
Ag.X
Ag.X
Agent PB
User Dir.
Rtg Table
US.CHISYS2
DTMU-A
..from MKT.PIAF at EUR.PARSYS1..to US.NYCSYS1;
OPS.CHASE, MFG.CHILD at US.CHISYS2..Dest Agent=X
DTMU-B
..from MKT.PIAF at EUR.PARSYS1..to OPS.CHASE,
..MFG.CHILD at US.CHISYS2...Dest Agent=X
Sample Network #2

Figure 14. Multiple-Destination Distribution with Intermediate Fan-out

Advantages of Using Intermediate DSUs

Reduced Connectivity Costs: The number of possible direct connections in a network is *(n(n-1))/2*, where *n* is the number of DSUs. If it is desired that direct sessions be used for all DS communication, this is the number of connections required.

DS networks could contain thousands of DSUs. If full interconnection is desired, millions of sessions would be required. Each DSU would need to have thousands of sessions. Only a small fraction would be active at any given time, but, nonetheless, the resources required would be unreasonably large.

Intermediate DSUs can be used to reduce this cost. If, instead of a fully interconnected network, all DSUs are connected to just one intermediate DSU, the total number of connections is *n-1*. When *n* is large, the savings in connection resources are dramatic. With 1000 DSUs the number of sessions needed is reduced from 499,500 to 999, a factor of 500 (i.e., *n/2*). In practice, rather than one intermediate DSU, a backbone network of 10 or 20 intermediate DSUs would be used and more sessions, perhaps another 200, would be needed. Even so, the resource savings are dramatic.

Improved Link Utilization: It is expected that some types of DS traffic will travel on sessions having lower transmission priority than the sessions that handle interactive traffic. Interactive loads fluctuate; the low-priority sessions serving DS traffic are expected to use whatever link capacity is available during lulls in the interactive loads.

When low-priority DS sessions span multiple links, the lulls in the interactive loads on all those links must be concurrent, or nearly so, for significant amounts of the DS traffic to flow. If one link spanned by the session is heavily utilized, it sets a limit on the throughput of the low-priority session over its entire length, and may prevent the low-priority traffic from using otherwise available capacity on the other links. Path control intermediate routing nodes (PCIRNs) have some buffering capacity, and can usually handle short delays caused by bursts of interactive traffic on a link. For longer delays, however, the PCIRNs may have to reduce the flow on the low-priority session.

The throughput between two DSUs may be increased by adding intermediate DSUs between them, so that low-priority DS sessions need not span so many links. Traffic is then able to flow over a particular session to an intermediate DSU while there is a lull on that session, even if the next "hop" to the destination happens to be busy because of interactive traffic. When there is a lull on the next "hop," the low-priority traffic can continue on its way.

In other words, the more links spanned by a low-priority session, the smaller is the probability that there will be concurrent lulls on all of them. If any link is fully utilized by high-priority sessions, the flow on the low-priority session is slowed to a trickle. The use of intermediate DSUs that provide a store-and-forward function may reduce transit times and increase throughput compared to direct connections.

Full-Function DS Networks

Distribution Service Parameters

DS is designed to provide a variety of types and levels of service. Originators may specify, as parameters on the verb by which they request a distribution, the levels of each type of service that the particular distribution requires. The values are included in the DS control information that flows through the network, and are used to condition the processing of the distribution at each DSU through which it flows. Each type of service requested is specified as a *service parameter.*

Service parameters may be used to map particular types of DS traffic to particular classes of service offered by the lower layers of SNA. They may be used to map different types of traffic to different routes through the DS network. At a particular DSU, the service parameters may be used to determine how to handle a distribution--for example, whether it must be safe-stored on nonvolatile storage.

Implementations are allowed to select, according to architecturally defined rules, those portions of the DS architecture that they implement. The subsetting rules for the architecture are defined in Appendix C. Certain implementation choices may result in networks in which the DSUs offer different levels of capability. The distribution service parameters are used to route distributions through only those DSUs capable of providing the requested service.

Service Parameters and Service Levels

DS allows the specification of many different service parameters. Certain parameters are defined by the architecture; others may be defined by particular implementations. Up to 10 different service parameters may be carried in one distribution.

Each service parameter is specified as a triplet. The triplet consists of

- the parameter type--architecturally defined parameter types are

  priority
  protection
  capacity
  security

- a comparison operator--the only comparison operator used by Format Set 2 implementations is REQUIRE_LEVEL_GE. Format Set 1 implementations use an additional comparison operator, REQUIRE_SUPPORT_FOR.
- a value--architecturally defined values vary by parameter type. They are discussed below.

The combination of comparison operator and value describes the level of service required by the distribution. For example, the originator might request a certain level of service (priority, perhaps) as a minimum, but be quite happy if the distribution traveled at a higher level. This request would be expressed as a comparison operator of REQUIRE_LEVEL_GE and a value indicating the minimum level acceptable.

The service parameters defined by the architecture are listed below, with a description of the comparison operators and values that may be specified for each.

The *priority* service parameter allows an originator to specify the relative urgency of a particular distribution. DS favors higher priority distributions when there is contention for resources. For example, when distributions are queued for sending, the higher priority ones are serviced first. In some cases, higher DS-priority distributions may flow on (path control layer) virtual routes with higher transmission priority. In general, the higher the DS priority, the more quickly the distribution flows through the network.

Any one of 16 different DATA priority values (DATA_1 through DATA_16, with DATA_16 being the highest priority) may be specified for ordinary distributions. Some implementations group DATA_1 through DATA_8 as DATALO and DATA_9 through DATA_16 as DATAHI. Above the DATA priority values are other values. In order of increasing priority, they are: CONTROL, which is used only for distribution reports (i.e., it may not be specified on an agent's request), and FAST, which is used for short, urgent types of messages.

The *protection* service parameter is used to specify whether the distribution must be stored on nonvolatile storage while a DSU has responsibility for it. One of two values may be specified--LEVEL1 or LEVEL2. LEVEL2 indicates that the distribution is safe-stored in nonvolatile storage; LEVEL1 indicates that safe-storage is not performed.

If the distribution needs to be protected in case of processor failure, the distribution request specifies *protection* (REQUIRE_LEVEL_GE LEVEL2). If the distribution does not require this level of protection, the request specifies *protection* (REQUIRE_LEVEL_GE LEVEL1).

An implementation may choose the level of protection it provides. The advantage of requesting a less demanding level of protection for a distribution is that there may be more DSUs capable of providing it, and therefore more routing possibilities. In addition, the processing time within a DSU might be decreased.

Some DSUs are able to handle only distributions whose DMU lengths are less than some specified maximum. The *capacity* service parameter allows DSUs with larger capacity to route large distributions around smaller-capacity DSUs that may not be able to handle them. Any route that passes through a size-limited DSU is appropriate only for distributions no larger than the maximum that the most limited DSU on the route can handle. This smallest maximum is the capacity of the route.

The values that an originator may specify for the capacity parameter indicate the minimum capacity of the DSUs through which the distribution should be routed. Often, all distributions generated by a particular agent specify the same capacity. If the distributions vary widely in size the agent would specify an appropriate capacity for each distribution.

The architecturally defined values are

ZERO
1MB (one megabyte)
4MB
16MB

The capacity parameter indicates the size of the server object contained in the distribution. Although a distribution with no server object might specify a capacity requirement of ZERO, the distribution could still contain an agent object.

The *security* service parameter is used to specify that the distribution is to be safeguarded from unauthorized access while it is being sent through the DS network. Two levels, LEVEL1 and LEVEL2, may be specified. LEVEL1 indicates that security is not required for the distribution. LEVEL2 indicates that DS should route the distribution only on sessions that are designated secure. (Typically, LU 6.2 session-level security would be used on such sessions.) When a distribution specifying LEVEL2 as the security value is stored at a particular DSU, the DSU and its general server (see "Servers and Objects" on page 40) ensure its security.

Default Service Levels

Any or all of the service parameters may be omitted from the DMU. The architecture defines a default service level for each parameter. A DSU processing a distribution uses the default service level for any parameter that is omitted from the DMU. The default values for each service parameter are given in Appendix G.

Combinations of Service Levels

The DS architecture allows a level to be specified for each service parameter, independent of the levels specified for other parameters. Applications may choose either to specify service parameters independently on a per-distribution basis or to routinely use certain combinations of parameters.

Uses of Distribution Service Parameters

From the time DS accepts responsibility for a distribution request until the distribution is delivered, the distribution service parameters may, and in some cases must, be used by all the DSUs to condition their processing of the distribution. The effects of the service parameters are apparent both in the routing of the distribution and the local handling of it.

Service Parameters in the Routing Table: Each entry in a DSU's routing table includes the levels of service that the route is able to provide. Each distribution carries the levels of service specified by the originator. When the distribution is routed, the requested levels are compared to the levels available.

To illustrate this process with a sample network, assume that a connection between EUR.PARSYS1 and US.CHISYS2 is to be reserved for small, high-priority traffic. Another connection, also reserved for small, high-priority traffic, is required between US.NYCSYS1 and US.CHISYS2. The resulting five-connection network appears as shown in Figure 15 on page 27. Notice that there are two connections between the same pair of DSUs, EUR.PARSYS1 and US.NYCSYS1. In order to distinguish between them, their identifiers must be qualified by a characterization of their service level capabilities. In this illustration, FAST and DATA are used.

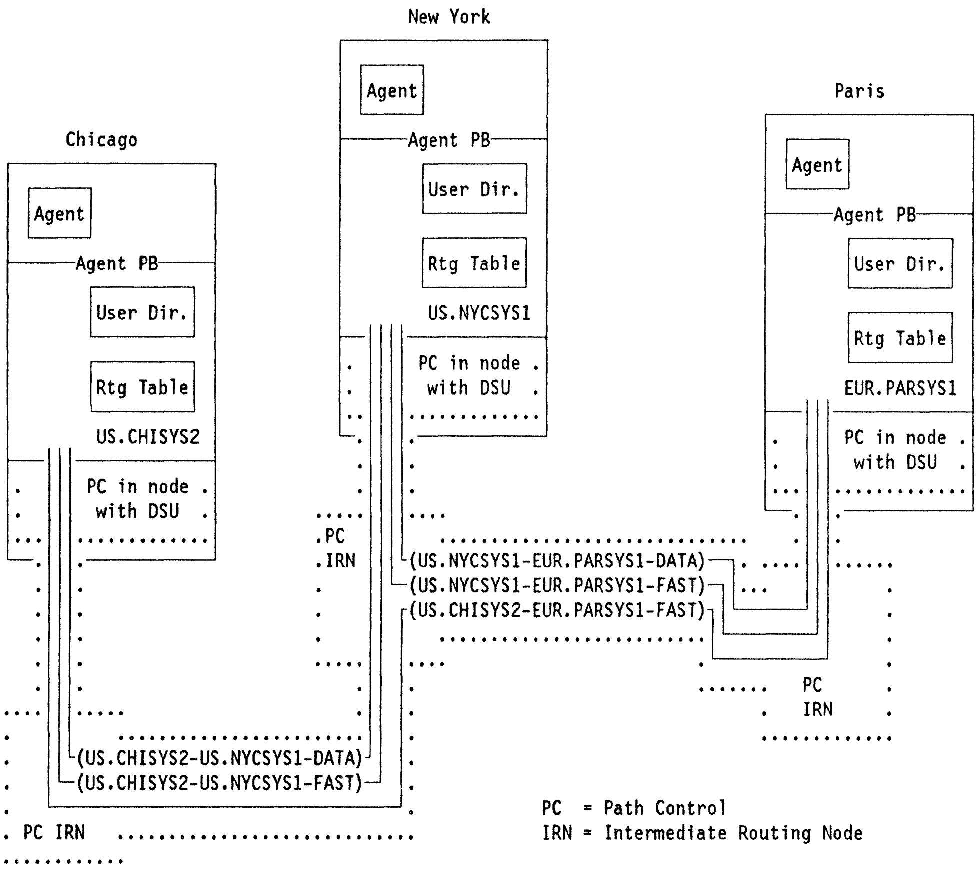
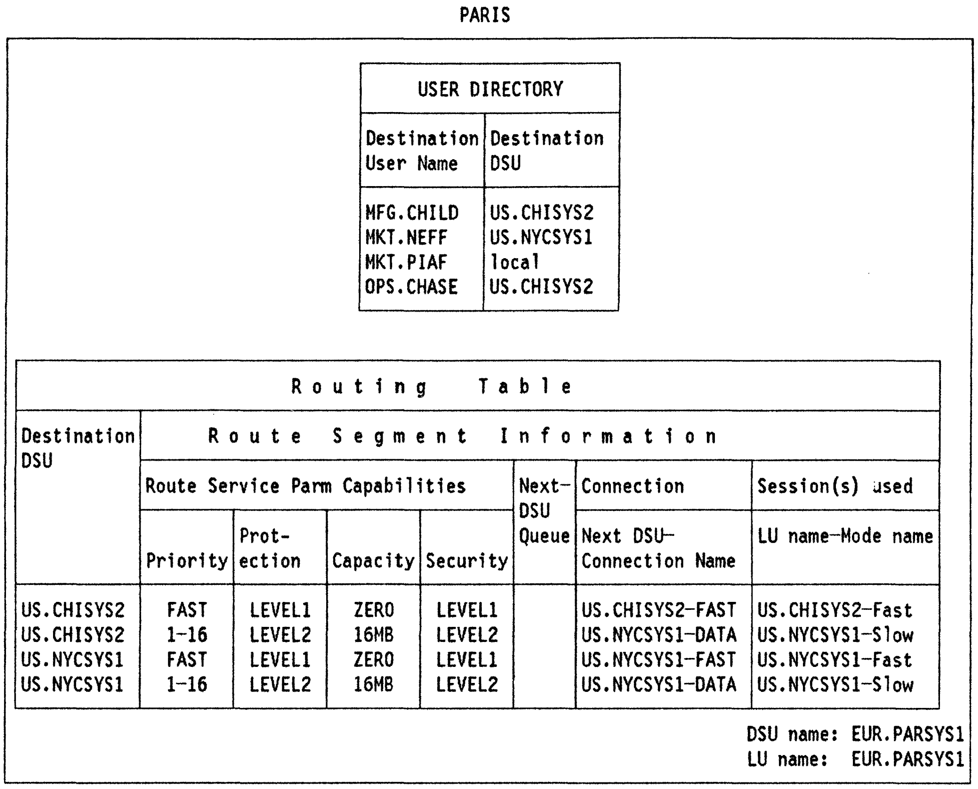

New York
Chicago
Paris
Agent
Agent PB
User Dir.
Rtg Table
US.NYCSYS1
US.CHISYS2
EUR.PARSYS1
PC in node with DSU
PC IRN
(US.NYCSYS1-EUR.PARSYS1-DATA)
(US.NYCSYS1-EUR.PARSYS1-FAST)
(US.CHISYS2-EUR.PARSYS1-FAST)
(US.CHISYS2-US.NYCSYS1-DATA)
(US.CHISYS2-US.NYCSYS1-FAST)
PC = Path Control
IRN = Intermediate Routing Node

Figure 15. Sample Network #3 Showing Different Routes for Different Priorities

The routing table in EUR.PARSYS1 contains entries as shown in Figure 16 on page 28. In addition to the connection identification of next-DSU name, the table entries include route service capabilities, queue identifiers, and the LU name and mode name (symbolized by "Fast" or "Slow") that identify the group of sessions used by the connection.

Each entry represents a route segment. The service capabilities describe the minimum capabilities that will be found along the route of which this particular route segment is part. Several route segments can share one queue. Several queues can share one connection. Generally, there is a one-to-one correspondence between a connection and the mode name of the session or group of sessions it uses.

When a distribution is to be routed, the combination of destination DSU and requested service parameters are used to scan the routing table in a serial fashion. The first entry for the destination DSU that provides acceptable service is used to identify the next-DSU queue into which the distribution is placed. After the distribution is placed in the next-DSU queue, a DS transaction program will retrieve it and send it on the connection to which that particular queue maps.



PARIS

USER DIRECTORY	
Destination User Name	Destination DSU
MFG.CHILD	US.CHISYS2
MKT.NEFF	US.NYCSYS1
MKT.PIAF	local
OPS.CHASE	US.CHISYS2

Routing Table							
Destination DSU	Route Segment Information						
	Route Service Parm Capabilities				Next-DSU Queue	Connection	Session(s) used
	Priority	Protection	Capacity	Security		Next DSU-Connection Name	LU name-Mode name
US.CHISYS2	FAST	LEVEL1	ZERO	LEVEL1		US.CHISYS2-FAST	US.CHISYS2-Fast
US.CHISYS2	1-16	LEVEL2	16MB	LEVEL2		US.NYCSYS1-DATA	US.NYCSYS1-Slow
US.NYCSYS1	FAST	LEVEL1	ZERO	LEVEL1		US.NYCSYS1-FAST	US.NYCSYS1-Fast
US.NYCSYS1	1-16	LEVEL2	16MB	LEVEL2		US.NYCSYS1-DATA	US.NYCSYS1-Slow

DSU name: EUR.PARSYS1
LU name: EUR.PARSYS1

Figure 16. The Paris DSU's Directory and Routing Table for Sample Network #3

These entries do not represent a formal definition of what is included in the routing table. Implementations arrange their table structures in whatever way is most appropriate for them. In particular, the network-unique DSU names and LU names shown in the illustration are not required. Local values such as pointers and offsets may be used.

The DSU name and the LU name need not be the same. For example, to facilitate network changes, it may be desirable to give one DSU multiple DSU names, but it would still have only one LU name. When the same value is used for the DSU name and LU name, installation and operations complexities are reduced.

Selecting Next DSU with Service Parameters: Notice the two entries for US.CHISYS2 in Figure 16. The first entry describes a route segment of a route with a priority capability of FAST connecting directly to the destination DSU. The second entry describes a route segment with a priority capability of DATA_1 through DATA_16 connecting to an intermediate DSU. If Piaf requests FAST for a single destination distribution to Chase in Chicago, the distribution is routed directly to US.CHISYS2. If, on the other hand, Piaf requests a priority between DATA_1 and DATA_16, the distribution is routed via the intermediate DSU US.NYCSYS1.

A multi-destination distribution sent from Piaf at EUR.PARSYS1 to US.NYCSYS1 and to Chase in Chicago is fanned out at the origin if priority FAST is specified, as shown earlier in Figure 9 on page 17. On the other hand, if priority DATA_4 is specified, the fan-out occurs at the intermediate DSU. This comparison illustrates how the use of direct sessions that bypass intermediate DSUs may shorten transit times but increase the amount of duplicate transmission. DS connections using sessions assigned to multi-link virtual routes should therefore be used sparingly, except for high-priority traffic. The larger the object being distributed, the greater is the benefit of intermediate fan-out.

Selecting a Session Using Service Parameters: In many cases, parallel (2 or more) sessions will exist between DSUs. The sessions will typically offer different classes of service. DS uses the LU 6.2 mode name to distinguish between the various kinds of sessions. For example, at the basic LU 6.2 layers of SNA, the session to US.NYCSYS1 with mode name "Fast" would be assigned a higher transmission priority than the session with mode name "Slow." DS would take advantage of the higher transmission priority by mapping a DS route of priority FAST to an LU 6.2 session with mode name "Fast."

Selecting Send Order Using Service Parameters: In some cases, only one session is available for a connection. Even when parallel sessions are available, it is generally desirable to limit the number of sessions used, particularly for low-priority, high-volume traffic. With limited numbers of sessions, in times of heavy loads on the network, the DS distributions will build up in queues for whatever next-DSU they must be sent to. These queues are called next-DSU queues.

The DS processes that access the routing table and put the distributions into next-DSU queues belong to the *routing* sublayer of DS. The processes that service the queues by sending the distributions out on the conversations belong to the *distribution transport* sublayer. The queues themselves can be viewed as straddling the protocol boundary of these two sublayers. Figure 17 on page 30 illustrates this.

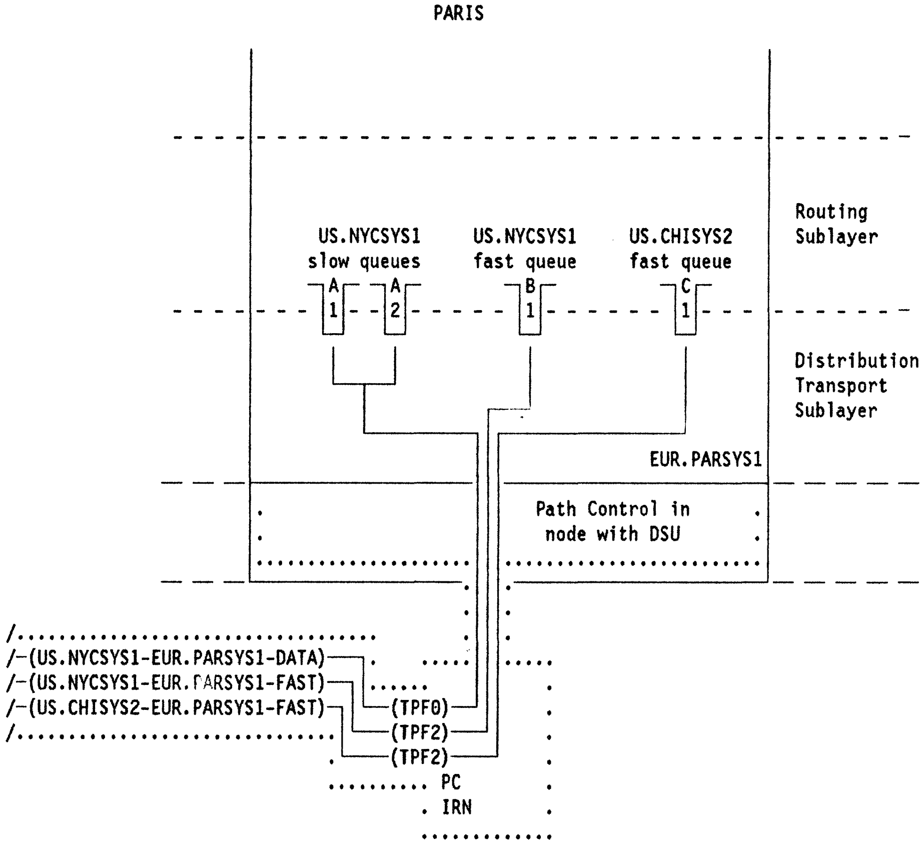


Figure 17. The Paris Section of Sample Network #3 Showing Two Queues for One Connection

Managing the next-DSU queues is an important part of the DS function. Requests specifying the same priority are queued on a first-in, first-out basis. Requests of higher priority may be assigned to the same sessions but are sent before those of lower priority.

One way this can be achieved is to maintain multiple queues for each connection and assign different priorities to separate queues. In the example shown in Figure 17, two queues exist for the DATA connection to US.NYCSYS1. The number suffix in the queue name represents the order in which the queues are serviced; that is, all the distributions in queue A1 are to be sent before any from queue A2. The routing table for EUR.PARSYS1 showing the next-DSU queue identifiers is given in Figure 18 on page 31.

PARIS

USER DIRECTORY	
Destination User Name	Destination DSU
MFG.CHILD	US.CHISYS2
MKT.NEFF	US.NYCSYS1
MKT.PIAF	local
OPS.CHASE	US.CHISYS2

Routing Table							
Destination DSU	Route Segment Information						
	Route Service Parm Capabilities				Next-DSU Queue	Connection	Session(s) used
	Priority	Prot-ection	Capacity	Security		Next DSU-Connection Name	LU name-Mode name
US.CHISYS2	FAST	LEVEL1	ZERO	LEVEL1	C1	US.CHISYS2-FAST	US.CHISYS2-Fast
US.CHISYS2	9-16	LEVEL2	16MB	LEVEL2	A1	US.NYCSYS1-DATA	US.NYCSYS1-Slow
US.CHISYS2	1-8	LEVEL2	16MB	LEVEL2	A2	US.NYCSYS1-DATA	US.NYCSYS1-Slow
US.NYCSYS1	FAST	LEVEL1	ZERO	LEVEL1	B1	US.NYCSYS1-FAST	US.NYCSYS1-Fast
US.NYCSYS1	9-16	LEVEL2	16MB	LEVEL2	A1	US.NYCSYS1-DATA	US.NYCSYS1-Slow
US.NYCSYS1	1-8	LEVEL2	16MB	LEVEL2	A2	US.NYCSYS1-DATA	US.NYCSYS1-Slow

DSU name: EUR.PARSYS1
LU name: EUR.PARSYS1

Figure 18. The Paris DSU's Routing Table with Two Queues for One Connection

The connection and session information shown in Figure 18 are not truly needed in the routing table. The queue name alone would suffice. The required relationships to connections and groups of sessions could be established by various additional tables and pointers. The example illustrates how distributions with DATA_N priorities flow over sessions with a mode name of "slow," which, in turn, can be assigned (at the path control level) to a virtual route with a transmission priority field (TPF) value of 0, the lowest of the three TPF values available.

Implementation Alternatives: The above example shows one way of ordering the sending by priority. Various implementations might choose other ways, such as maintaining one queue but keeping the entries in it sorted by priority.

Some implementations delay the decision of which connection to use for a distribution. Distributions can be queued by routing table entry. The determination of which connection should be used to service which queue can be deferred. In other words, the static relationships depicted in the routing tables illustrated here would be partly replaced by a dynamically determined relationship. The determination could be triggered by some event or condition. For dial-up networks, the determination might be triggered by the activation of the connection.

Local Handling of Distributions: Service parameters are also used to condition the DSU's local handling of distributions. For example, if a distribution request specifies *protection* (REQUIRE_LEVEL_GE LEVEL2), the DSU is required to store it in nonvolatile storage before accepting responsibility for it. If a distribution request specifies *security* (REQUIRE_LEVEL_GE LEVEL2), the DSU may use additional parameters on the commands it issues to the server to access the distribution (see "Servers and Objects" on page 40 for a discussion of servers). Complete information on the actions taken by DSUs in response to specified service parameters is given in Appendix C.

Sublayering in DS Networks

DS is modeled as part of the transaction services layer of SNA, but is itself composed of sublayers. The notion of the distribution transport and routing sublayers was introduced in Figure 17 on page 30. Two more DS sublayers are defined: an uppermost sublayer that handles requests from agents, called the DS presentation sublayer, and a sublayer between DS presentation and routing that provides the directing process. Hence, DS consists of four sublayers existing network wide, each sublayer existing in every DSU. The sublayers are:

DS Presentation
Directing
Routing
Distribution Transport

Most implementations of DS employ queues at the boundaries between the presentation and directing sublayers, and between the routing and transport sublayers. There is less need for queuing between directing and routing, and it is not considered in this document.

Processes Performed in the DS Sublayers

The processes that perform the DS functions can be placed in, or in some cases across, the sublayers. Each DSU contains instances of the various processes, provided by a collection of service transaction programs. A detailed model of the internal structure of a DSU is given in Chapter 3.

- The processes in the DS presentation sublayer are called DS presentation services (PS).[1] Agents interact with PS to use DS services. The following interactions are supported:
  - A Send_Distribution verb initiates a distribution to one or more destinations. PS validates the request before accepting it. If PS accepts the request, it enqueues the distribution for processes in the directing sublayer.
  - A Receive_Distribution verb provides the destination agent with access to the distribution.

[1] The DS presentation services (PS) sublayer is distinct from the SNA layer below the transaction services layer (in which DS resides), which is also called presentation services. The latter layer processes the LU 6.2 basic conversation verbs issued in DS, whereas DS PS processes DS verbs issued by application transaction programs. In this book, "PS," unqualified, generally refers to the DS sublayer.

Other variations of the basic send and receive verbs provide agents with additional capabilities. The other verbs are discussed in "Agent Protocol Boundary Verbs" on page 55 and Appendix F.

  - Operations verbs provide access to enqueued distributions or to DS resources, such as the routing table, to display or change them. Operations verbs also provide access to logged exception information. The operations verbs are described in "Operations" on page 68 and Appendix F.

Verbs and parameters are passed across the protocol boundary to DS. Return codes are passed back indicating that the function has been completed, successfully or not, as indicated by the code.

- The processes in the directing sublayer perform the following functions:

  - Determine the destination DSU names corresponding to destination user names and add them to the distribution--this is done at the origin of the distribution.

  - Determine the local queues and application program identifiers for distributions at their destinations.

  - Change the destination DSU names for users whose destination DSU is the local one, but who are not, in fact, local users--the redirection case (see "Redirection" on page 35).

  Directing then invokes routing for nonlocal destinations.

  When a distribution is to be placed on more than one local queue, it is "fanned out" by directing.

- The processes in the routing sublayer perform the following functions:

  - Identify inbound distributions for which at least one destination DSU name is local. Directing is invoked for these destinations.

  - Use the routing table to select appropriate next-DSU queues for outbound distributions. Routing may perform "fan-out" in the case where the distribution is placed on more than one queue.

- The processes in the distribution transport sublayer manage the communication between DSUs to send a distribution from one DSU to the next. The queues into which the routing processes place distributions are accessed in predetermined order. This sublayer performs the encoding of the distribution into a distribution message unit and handles the DS protocol for the transmission of the message unit. Protocols to exchange exception information are handled by this sublayer as well. LU 6.2 basic conversation verbs are issued and return codes and parameters are analyzed to accomplish the transmission.

  The processes in the distribution transport sublayer invoke a facility known as a *server* to gain access to the *server object* when sending, and to store the received server object when receiving. The concepts of servers and server objects are explained in "Servers and Objects" on page 40.

  Processes in this sublayer manage the transfer of responsibility for a DMU. The sending DSU is responsible until the receiving DSU has signaled via a DS protocol that it has accepted responsibility.

Sublayer Diagrams

All the sublayers of DS exist as part of the transaction services layer of SNA. The upper edge of DS is the DS agent protocol boundary (agent PB). This PB makes up only a part of the total PB offered to users by SNA services. The lower edge of DS is the basic conversation protocol boundary of LU 6.2.

DS uses the basic conversation protocol boundary of LU 6.2 for sending and receiving its DMUs. DS shares the use of this PB with other service transaction programs. In the following diagrams, that protocol boundary is depicted as a solid line labeled LU 6.2 BCPB. It should not be confused with the mapped conversation protocol boundary used by application transaction programs. A more detailed illustration of the structure of a DSU can be found in Chapter 3.

As an example, once again assume that Piaf wants to send a low-priority distribution (single destination) to Chase. The distribution is sent through US.NYCSYS1, where only the routing function is involved. From a sublayering standpoint, the distribution is handled as shown in Figure 19.

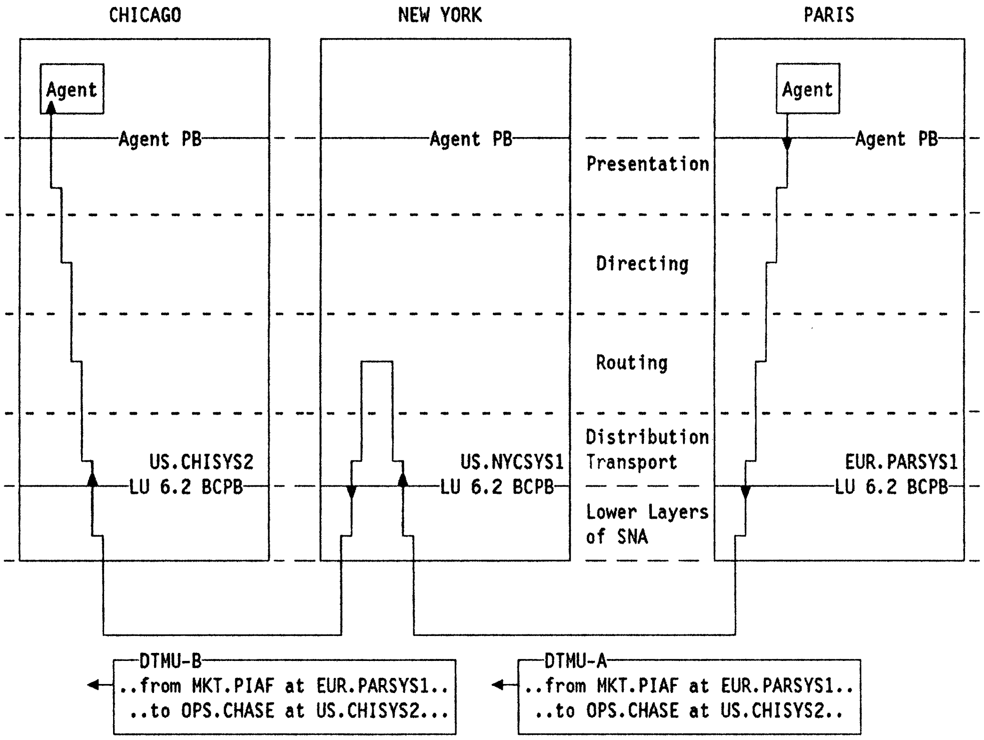


Figure 19. DS Sublayering Diagram--Intermediate Routing

Redirection

Looking once again at sample network #2, particularly the directories in Figure 12 on page 20, assume that Neff in marketing is transferred to Chicago for a brief assignment, not long enough for the network administrators to want to change all the directories in the network. The *redirection* capability of the DSU in New York simplifies the required directory changes. Only the directories at Neff's old and new locations need be changed. Refer to Figure 20 and contrast the directory entries for MKT.NEFF with those in Figure 12 on page 20.

CHICAGO

USER DIRECTORY

Destination User Name	Destination DSU
MFG.CHILD	local
MKT.NEFF	local
MKT.PIAF	EUR.PARSYS1
OPS.CHASE	local

ROUTING TABLE

Destination DSU	Connection (Next DSU)
EUR.PARSYS1	US.NYCSYS1
US.NYCSYS1	US.NYCSYS1

US.CHISYS2

NEW YORK

USER DIRECTORY

Destination User Name	Destination DSU
MFG.CHILD	US.CHISYS2
MKT.NEFF	US.CHISYS2
MKT.PIAF	EUR.PARSYS1
OPS.CHASE	US.CHISYS2

ROUTING TABLE

Destination DSU	Connection (Next DSU)
EUR.PARSYS1	EUR.PARSYS1
US.CHISYS2	US.CHISYS2

US.NYCSYS1

PARIS

USER DIRECTORY

Destination User Name	Destination DSU
MFG.CHILD	US.CHISYS2
MKT.NEFF	US.NYCSYS1
MKT.PIAF	local
OPS.CHASE	US.CHISYS2

ROUTING TABLE

Destination DSU	Connection (Next DSU)
US.CHISYS2	US.NYCSYS1
US.NYCSYS1	US.NYCSYS1

EUR.PARSYS1

Figure 20. Directories Illustrating Temporary Redirection

Suppose that Piaf in Paris sends a distribution to MKT.NEFF. The DSU in Paris associates the destination DSU US.NYCSYS1 with MKT.NEFF. When DTMU-A arrives at US.NYCSYS1, the routing function there recognizes its own name, presumes that it is the destination DSU, and passes the distribution up to the directing sublayer. The directing function accesses the user directory and finds, instead of local delivery information, another DSU name. It then replaces US.NYCSYS1 with the new name, US.CHISYS2, and passes the modified control information back down to routing. Routing then queues the distribution for US.CHISYS2.

From a sublayering standpoint, the distribution is handled as shown in Figure 21 on page 36. The routing process invokes the directing process because the destination DSU name is local. Directing, after determining that the destination user is not local and after changing the destination DSU name, invokes routing.

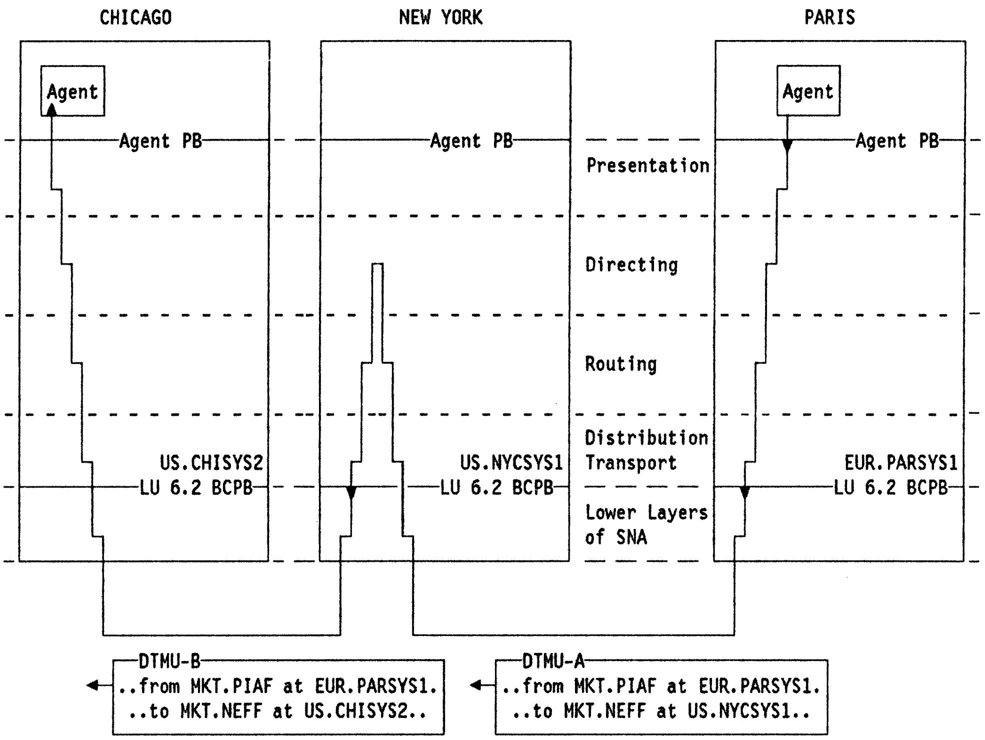


Figure 21. DS Sublayering Diagram--Redirecting

The redirection processing at the intermediate DSU, US.NYCSYS1, should be compared to Figure 19 on page 34, which depicts pure routing at the intermediate DSU.

In the example above, the redirection occurred at the most convenient point. Suppose, however, that Chase had moved to New York. A message from Piaf to him would be redirected at US.CHISYS2 back to US.NYCSYS1, the DSU through which it had just been routed. In networks of reasonable complexity, some occurrences of redirection may result in routing inefficiency. System administrators must use care to keep this to a minimum.

A DSU refers to its directory for any distribution whose destination list includes that DSU's name. A DSU also refers to its directory if the destination list includes any of the DSU names in the DSU's intervention list (see "The Intervention List" on page 40). In either case, redirection may result.

Default Directing

In large networks, the number of users makes it impractical to have an explicit entry for every user in every DSU's directory. Instead, default "tokens" (the "*" in this documentation) can be used in place of parts of the user name; that is, either the DEN, or both the DGN and the DEN. The default token is like a wild card that can assume any value. User names for which a complete explicit match (that is, an exact match on both DGN and DEN) cannot be found are matched against the entries with explicit DGNs and * for DENs. User names that fail to match any DGN.* entry must by definition match the *.* entry, which is simply the equivalent of "unable to find any match." In other words, assuming that the directory is in collating sequence and the search algorithm is serial, the default tokens mean "none of the preceding" matched.

Whatever DSU name is found at a default entry is used exactly the same way as the DSU name found at an explicit entry. That is, it is associated with the user name and is used for routing through the network. In some cases, the DSU name assigned by default will prove to be correct. In other cases, redirection will occur at the default DSU. In properly defined networks, the directories at the default DSUs would have more explicit entries and, therefore, would be better able to direct distributions explicitly--that is, to direct them to the DSU where the user really is.

The sample networks discussed earlier in this chapter have all had only four users, an extreme simplification employed to minimize table sizes in the examples. To illustrate default directing, it is helpful to imagine a network with the same three DSUs, but serving more users. This network is only slightly larger than sample network #1. The following example is a complete network and not a fragment of a larger one.

```
o  CHICAGO
Manufacturing Dept.
   Child
Marketing Dept.
   Chapin
Operations Dept.
   Chase
   Chalk
```

USER DIRECTORY

Destination User Name	Destination DSU
MFG.CHILD	local
MFG.*	US.NYCSYS1
MKT.CHAPIN	local
MKT.NEFF	US.NYCSYS1
MKT.PIAF	EUR.PARSYS1
MKT.*	US.NYCSYS1
OPS.CHASE	local
OPS.CHALK	local
OPS.PLACE	EUR.PARSYS1
OPS.*	error
.	US.NYCSYS1

ROUTING TABLE

Destination DSU	Connection (Next DSU)
EUR.PARSYS1	US.NYCSYS1
US.NYCSYS1	US.NYCSYS1

US.CHISYS2

```
o  NEW YORK
Manufacturing Dept.
   North
Marketing Dept.
   Neff
   Nelson
   Nesbit
```

USER DIRECTORY

Destination User Name	Destination DSU
MFG.CHILD	US.CHISYS2
MFG.NORTH	local
MFG.*	error
MKT.CHAPIN	US.CHISYS2
MKT.NEFF	local
MKT.NELSON	local
MKT NESBIT	local
MKT.PIAF	EUR.PARSYS1
MKT.*	error
OPS.PLACE	EUR.PARSYS1
OPS.*	US.CHISYS2
.	error

ROUTING TABLE

Destination DSU	Connection (Next DSU)
EUR.PARSYS1	EUR.PARSYS1
US.CHISYS2	US.CHISYS2

US.NYCSYS1

```
o  PARIS
Marketing Dept.
   Piaf
Operations Dept.
   Place
```

USER DIRECTORY

Destination User Name	Destination DSU
MKT.NEFF	US.NYCSYS1
MKT.NESBIT	US.NYCSYS1
MKT.PIAF	local
MKT.*	US.NYCSYS1
OPS.CHASE	US.CHISYS2
OPS.PLACE	local
OPS.*	US.CHISYS2
.	US.NYCSYS1

ROUTING TABLE

Destination DSU	Connection (Next DSU)
US.CHISYS2	US.NYCSYS1
US.NYCSYS1	US.NYCSYS1

EUR.PARSYS1

Figure 22. Sample Network #4--Directories with Default Entries

Figure 22 depicts the directories for sample network #4. The directory at EUR.PARSYS1 contains default DEN entries for two DGNs, MKT and OPS, pointing to US.NYCSYS1 and US.CHISYS2, respectively. The directory at US.NYCSYS1 contains default DEN entries for all three DGNs. At US.NYCSYS1, however, MFG.* and MKT.* are errors. This means that the lists of explicit entries for those DGNs are exhaustive. If a DEN other than those listed occurs, it must be in error. The OPS.* points to US.CHISYS2 where the directory for OPS is exhaustive. In this example, therefore, there is a complete set of DENs for every DGN in at least one DSU's directory.

The directory at EUR.PARSYS1 contains no explicit entry for the DGN MFG. Any user names beginning with MFG (or any DGN other than MKT or OPS) will match the *.* entry at the bottom of the directory and be directed to US.NYCSYS1. The directory at EUR.PARSYS1 will never detect an invalid user name. The directory at US.CHISYS2 will detect invalid DENs within the OPS DGN but any unrecognized DGNs will be directed to US.NYCSYS1. Because the directory at US.NYCSYS1 is the only one with a *.* error entry, it is the only place at which completely invalid user names will be detected.

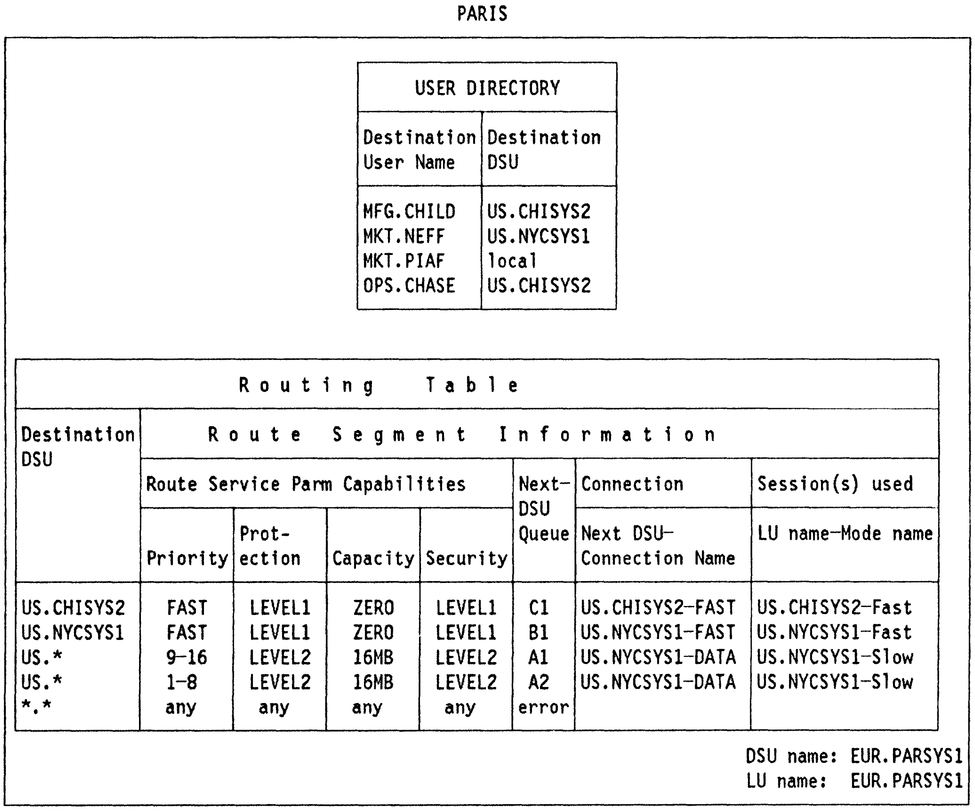
Customers should define their networks so that every DGN has a complete set of explicit entries--that is, every DEN for that DGN--in a directory somewhere in the network. Similarly, at least one directory should contain a complete set of DGNs. Also, every DSU's directory should be complete in the sense that any valid user name not explicitly matched will be redirected. Thus, the service can distribute from any DS-defined user to any other DS-defined user, always detecting invalid user names (for example, misspelled names) as such.

Note that a *.* default entry is different from an entry for omitted user names (as shown in Figure 7 on page 15). The latter matches destinations for which no user name was specified in the distribution request (i.e., node destinations). The former matches user destinations (those for which a user name was specified in the request) that cannot be found in the local directory.

Default Routing

In large networks, it is also impractical to maintain routing tables in every DSU with explicit entries for every other DSU. Default tokens can be used in the routing table in exactly the same way as in the directory. It would not be an error or even unusual to have an explicit entry in the directory that returned a destination DSU for which no explicit entry exists in the routing table.

Looking again at sample network #3, Figure 17 on page 30, and the routing table in Figure 18 on page 31, it can be seen that the routing table has six entries, although EUR.PARSYS1 has only four queues for three sessions. The routing table could be shortened by the use of default routing tokens as shown in Figure 23 on page 39.



PARIS

USER DIRECTORY	
Destination User Name	Destination DSU
MFG.CHILD	US.CHISYS2
MKT.NEFF	US.NYCSYS1
MKT.PIAF	local
OPS.CHASE	US.CHISYS2

Routing Table							
Destination DSU	Route Segment Information						
	Route Service Parm Capabilities				Next-DSU Queue	Connection	Session(s) used
	Priority	Prot-ection	Capacity	Security		Next DSU-Connection Name	LU name-Mode name
US.CHISYS2	FAST	LEVEL1	ZERO	LEVEL1	C1	US.CHISYS2-FAST	US.CHISYS2-Fast
US.NYCSYS1	FAST	LEVEL1	ZERO	LEVEL1	B1	US.NYCSYS1-FAST	US.NYCSYS1-Fast
US.*	9-16	LEVEL2	16MB	LEVEL2	A1	US.NYCSYS1-DATA	US.NYCSYS1-Slow
US.*	1-8	LEVEL2	16MB	LEVEL2	A2	US.NYCSYS1-DATA	US.NYCSYS1-Slow
.	any	any	any	any	error		

DSU name: EUR.PARSYS1
LU name: EUR.PARSYS1

Figure 23. Routing Table with Default Entries

Notice that there is an explicit entry for fast priority traffic to US.CHISYS2, but no other explicit entries for US.CHISYS2. The US.* entries will cause any distribution with an unrecognized REN (the low order part of the DSU name), and a priority of DATA_1 to DATA_16 to be routed to US.NYCSYS1. The *.* entry catches any unrecognized DSU names. Since this routing table is complete (i.e., any distribution for either of the other DSUs in the network will be handled by one of the entries), the *.* entry maps to an error condition. Alternatively, the *.* entry could be used to route all distributions destined for unrecognized DSU names elsewhere, just as a *.* entry in the directory causes distributions for unrecognized user names to be directed elsewhere.

Network administrators should ensure that every routing table is complete in the sense that it contains either an explicit match or a default match for every destination DSU.

Alternate Routing

Implementations may provide a mechanism to perform alternate routing when connections are unavailable. For example, the routing table might contain two entries for the same destination DSU and service parameter combination, with each entry identifying a different next-DSU. Assuming that the routing table is scanned in a serial fashion, the first entry would normally be matched, and all distributions would be sent to that next-DSU. If the connection to that next-DSU became unavailable, the DSU could mark its routing table entry unavailable.

The routing lookup process could then bypass that entry, and subsequent distributions would be routed to the next-DSU named in the second entry. Since the specific contents of an implementation's routing table are not defined by DS, such a mechanism is outside the scope of the architecture.

The Intervention List

A DSU may intervene in traffic destined for other DSUs. The *intervention list* specifies the names of other DSUs for which the DSU is to process traffic. These DSU names may or may not be the names of other DSUs in the network. A DSU name may appear in the intervention list of one or more DSUs.

A DSU checks the intervention list when a distribution is received. The DSU invokes the directing process for traffic addressed to any of the DSU names in its intervention list, just as it does for traffic addressed to its real name. This feature can be used to facilitate network rearrangements. For example, when DSUs are combined, the resulting DSU would have the names of the eliminated DSUs in its intervention list.

Another use of the intervention list is to facilitate "Big Brother/Little Brother" implementations. For example, suppose that a particular implementation has only very limited resources to allocate to a directory and routing table, and that it supports only a small number of local users. The directory can be set up to define only the local users, and to route traffic (by default) for all other users to a partner DSU whose directory contains information for remote users.

Since its directory would not contain information for remote users, redirection performed by the smaller DSU would be inefficient. An effective solution is to define the small implementation in the intervention list of the adjacent partner. The partner then processes all traffic destined for the smaller DSU as it does its own; the partner can perform redirection as necessary before sending traffic to the smaller DSU.

Servers and Objects

Introduction

Large amounts of data that are to be distributed through the DS network are typically accessed via application programs called *servers*. Such data is encoded in the DTMU as the *server object*.

When an agent issues a distribution request that is to contain a server object, it normally does not include the object itself in the request, but rather points to where the object can be found. The pointer to the object is stored by the DSU as part of the control information for the distribution. When the DSU is ready to send the distribution, it uses the server to read the object, one piece at a time. As each piece is retrieved, the DSU issues LU 6.2 basic conversation verbs to feed it into the network. The partner DSU receives the object, one piece at a time, and uses a server to store it away. The origin agent is responsible for specifying a server appropriate for the object to be distributed.

Server Names: The name of the server to be used at the origin and destination DSUs is specified as part of the request when the request refers to a server object. Server names are taken from the same name space as agent names.

DS is unaware of an implementation's internal packaging of servers. A single piece of implementation structure might be able to responsibly store and retrieve objects for a wide variety of server names. From the DS perspective, that single piece would be viewed as multiple servers.

Origin Servers: DS uses the origin server to access the object for sending. No direct interaction takes place between the requesting agent and the origin server it specifies, either when the request is made or when the DTMU is sent. (The requesting agent stores the object using the origin server before issuing a distribution request to DS.) DS acts as the intermediary while processing the distribution. DS is unaware of interactions that take place between agents and servers when distribution requests are not involved.

Destination Servers: The server name is put into the DTMU and flows through the network to all the destinations. Only one server name exists for the server object, no matter how many destinations are specified. The destination server is used at the destination DSU to store the object in application space (for example, a library shared by several users).

Server Implementations: Servers may be of a unique kind implemented as part of the same application as the agent, or they may be general-purpose servers included in the supporting environment.

Reversible Servers: A reversible server is one that can be invoked either for storing or retrieving. When instructed to retrieve an object that it has previously stored, it retrieves the same byte stream it stored earlier. This does not mean that the byte stream must actually be stored in a byte-stream image, but it does mean that the server is able to reverse any transformations it might have performed upon the byte stream. Typically, servers would be reversible. An example of a nonreversible server would be one that printed hard copy.

A DSU invokes a nonreversible server only if it has determined that the object will never need to be retrieved for forwarding or delivering multiple copies.

Use of Servers: Figure 24 on page 42 gives a simple illustration of the interactions among the agent, server, and DSU. At some point before issuing a distribution request, the agent interacts with the server to store an object (arrow 1). The agent then issues a request to DS, specifying the server name (arrow 2). When a connection is available to the next-DSU, DS invokes the server to read the server object (arrow 3), builds the DTMU, and sends it.

The adjacent DSU receives the DTMU from its partner, invokes a server to store the object (arrow 4), and subsequently starts the destination agent. When the agent receives the distribution, which includes the destination server information (arrow 5), DS's responsibility for the distribution is fulfilled. The agent may use the server information to access the object at a later time (arrow 6).

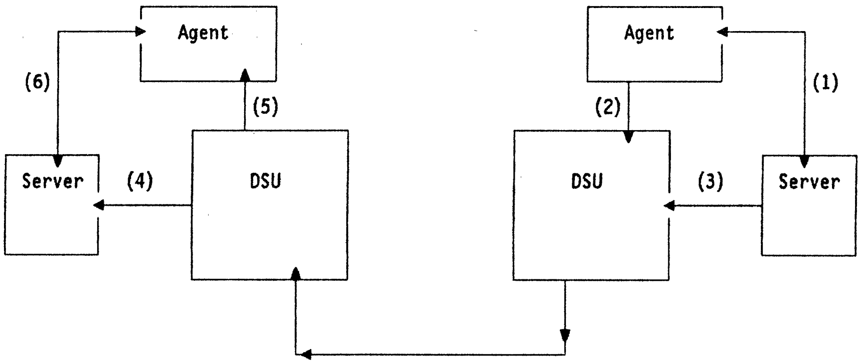


Figure 24. Simple Agent-Server-DSU Interaction

DS considers the server object to be the byte stream passed across the server protocol boundary via the Read and Write verbs. DS numbers the bytes in this stream beginning at 1. A server object may be presented to the user or stored in safe storage in a form very different from that seen by DS, but these other forms are irrelevant to DS. For example, a 10-character text string might be compressed into six bytes when stored on disk, and encrypted into a 20-byte string for transmission. DS knows nothing of the 10-character text message, or the 6-byte compressed form stored on disk; it knows only about a 20-byte server object. DS numbers the bytes in this object from 1 to 20.

General and Specific Servers

Specific servers store and retrieve objects into and from users′ or applications′ private space. They are so named because they often perform some type of application-specific handling of the objects on which they operate; for example, a specific server might provide encryption and decryption of data, or it might perform specialized parsing of the objects passed to it. A specific server may be closely affiliated with one particular agent that uses it, or it may provide services to several agents.

Specific servers may be sensitive to the contents of the byte stream passed to them. They may reject the object because it violates application-specific rules. They are not expected to cope with byte streams other than those defined for them.

The *general server* is a server that DS uses to store server objects in DS′s storage space. It is a reversible server that is completely insensitive to the contents of the objects it stores.

DS perceives the objects it transports as no more than streams of bytes. Whenever a DSU is uncertain as to whether or not an object will have to be retrieved for forwarding or creating additional local copies, the DSU stores the object with a reversible server. If the DSU has available to it the destination server named in the distribution, and if that server is reversible, the DSU may use it to store the object without having to determine its role for the distribution.

In general, however, there will be distributions received with destination server names that the DSU does not have. (This implies that the DSU's role for the distribution in question is purely intermediate, since it could not make any local deliveries.) In such cases, the DSU invokes a general server, totally ignoring the destination server name. In some implementations, the program invoked might be the same as that used to provide a specific server function. The name specified in the server protocol boundary verbs, however, would be that of the general server. To DS, therefore, it would be a different server.

The general server is not sensitive to the contents of the byte stream passed to it. It cannot reject an object because of its contents. Typically, the space into which a general server stores its objects is associated with the system rather than a particular user or application.

The general server's name is used in server protocol boundary verbs but usually does not flow in DTMUs. The server name specified in the distribution request (which then flows in the DTMU) is usually a specific server name. The originating agent might, however, choose to specify the general server name for objects that have no significant internal structure and can conveniently be accessed by the destination agent as a serial byte stream. This implies that the space into which the general server stores objects is accessible to the origin and destination agents.

The most general model of DS's use of servers is illustrated in Figure 25 on page 44. In this model, the agent's request (arrow 1) names a specific server to be used at the origin and destination. (Arrow 0 indicates the agent's interaction with the server before invoking DS.) The DSU copies the server object from the user's (or agent's) private space (arrow 2) into system space (arrow 3), invoking the origin specific server to read the object and the general server to write it. Depending on parameters specified on the agent's request, the DSU may or may not make this copy before returning control to the agent. Once the object has been copied into DS space by the general server, DS has responsibility for the distribution request. When a connection is available to send the distribution to the adjacent DSU, DS invokes the general server to read the object (arrow 4), builds the object into the DTMU, and sends it to the partner DSU.

The partner DSU, upon receiving the DTMU, stores the server object into system (DS) storage space using the general server (arrow 5). If the DSU discovers, during the routing and directing process, that the distribution contains destinations local to the DSU, it copies the server object into the destination user's (or agent's) private storage space, invoking the general server to read the object (arrow 6) and the destination specific server (named in the distribution) to write it (arrow 7). The destination server information is passed to the destination agent when it receives the distribution (arrow 8). The agent may then access the object using the specific server at a later time (arrow 9).

At an intermediate node, the DSU receives the DTMU and stores the object using its general server. When the distribution is to be sent to the next DSU, the general server is invoked to retrieve the object for sending (Figure 26 on page 45). Since the DSU is not a destination for the distribution, the object is not copied to the specific server.

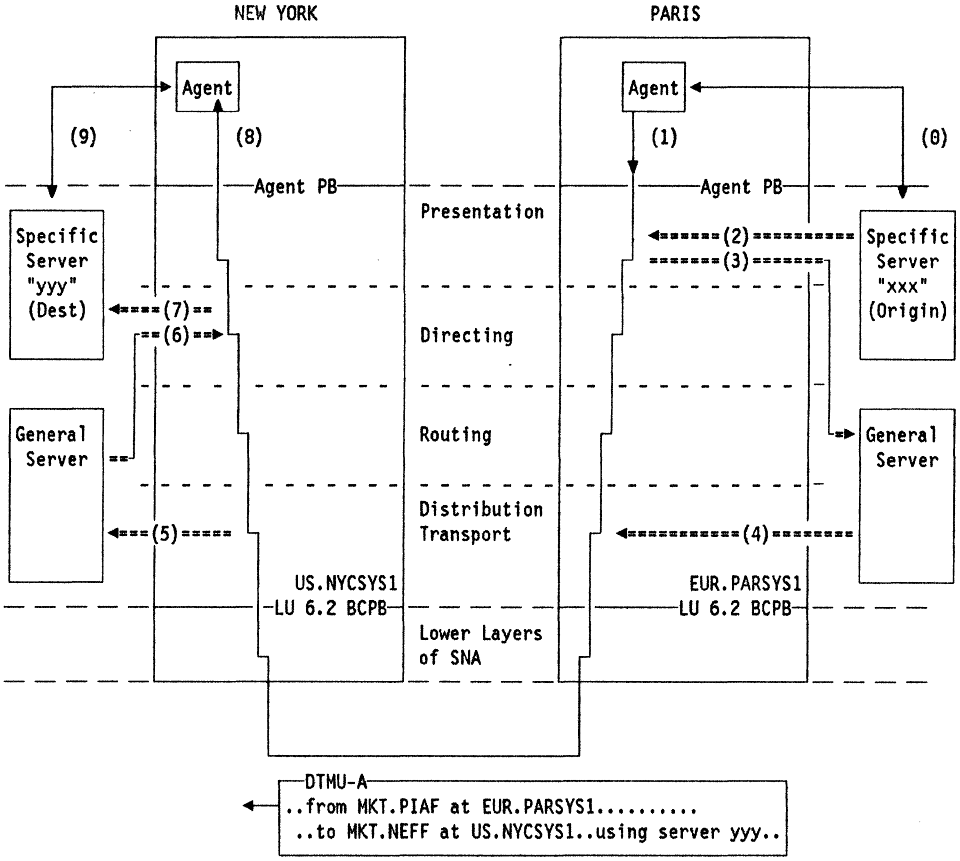


Figure 25. DS's Use of Servers--General Model

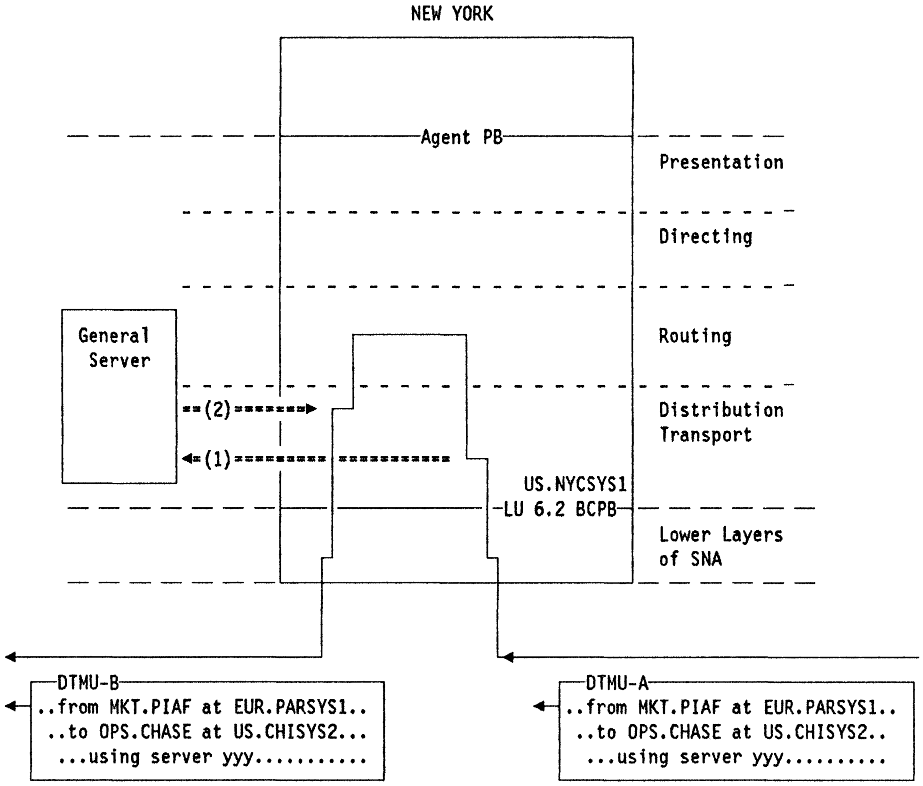

NEW YORK
Agent PB
Presentation
Directing
Routing
General Server
Distribution Transport
(2)
(1)
US.NYCSYS1
LU 6.2 BCPB
Lower Layers of SNA
DTMU-B
..from MKT.PIAF at EUR.PARSYS1..
..to OPS.CHASE at US.CHISYS2...
...using server yyy..........
DTMU-A
..from MKT.PIAF at EUR.PARSYS1..
..to OPS.CHASE at US.CHISYS2..
...using server yyy.........

Figure 26. The General Server at an Intermediate DSU

Server Exceptions and Reporting

Errors or exceptions may occur during any of the server operations that are performed as part of a distribution. The actions that DS takes and the exception reports that it generates vary, depending on which server detects the exception. Specific servers are viewed as belonging to a using application (as are agents); the general server is viewed as belonging to DS. DS's actions in response to an exception indication from a server reflect this notion.

The most general model of DS's use of servers is illustrated in Figure 25 on page 44. At the origin, the server object is copied from the origin specific server to the general server. At the destination, the object is copied from the general server to the destination specific server. These copy-making steps are referred to as *auxiliary server operations.*

Auxiliary server operations are performed by DS as a service to the application program, prior to accepting responsibility for the request at the origin and after receiving the distribution at the destination. An exception that occurs during an auxiliary operation is reported via either a distribution report or a local server report, depending on whether the general or the specific server detects the exception.

An exception that involves DS or the general server is reported via a distribution report. An exception that involves the specific server is reported to the local agent via a local server report.

At the origin, an exception detected by the specific server results in the delivery of a local server report to the local agent. The distribution is aborted (it could not be accepted by DS, since the server object could not be read).

Once the object has been successfully copied to the general server at the origin and DS has accepted responsibility, any exceptions detected by the general server or DS are reported via distribution reports. Whether at an intermediate DSU or at the destination DSU, if the general server is unable to store the object upon receipt of the distribution, DS generates a report and sends it to the "report-to" destination specified by the originator (see "Distribution Reporting" on page 66).

After the distribution has been received by the destination DSU and the server object has been successfully stored by the general server, an auxiliary operation is performed to copy the object from the general server to the specific server (arrows 6 and 7 in Figure 25 on page 44). An exception during this operation is treated similarly to a failure during the auxiliary operation at the origin. If the exception involves DS or the general server, the exception is reported via a distribution report, and the distribution is terminated. If the exception involves the specific server, it is reported via a local server report to the local agent, and the distribution is delivered to the destination agent.

The rules for reporting server exceptions reflect the notion that the specific server belongs to the application and the general server belongs to DS. Since a successful general-server operation means that DS has been able to transport the object through the network to the destination, a subsequent specific-server exception during the auxiliary server operation is deemed to be the responsibility of the application; it is thus reported locally to the destination agent on the Receive_Distribution verb.

Early Acceptance of the Server Object

The essence of the reporting rules described above is that, at both the origin and the destination, specific servers report exceptions to their agents, through DS. A specific server could have a variety of reasons for reporting to its agent. Even if the auxiliary operation is completely successful, a report might be required.

A more general case is that of a partially successful operation. For example, the origin server might be unable to find all the items listed in the distribution request. If the origin server decides to send the distribution despite the missing items, it simply indicates to DS a normal completion of the server operation. Only the server would know that the DTMU contains less than the Send_Distribution verb requested. A responsible origin server would then report its partial failure to the origin agent via a local server report. Such a report has nothing to do with DS, except that DS delivers it to the agent across the agent protocol boundary.

At the destination as well, the specific server may encounter a partially successful operation. It may then choose whether or not to continue receiving the object.

If it chooses to continue the operation, perhaps discarding whatever portions of the server object it cannot handle, DS is unaware of any abnormality. Only the server knows that the original Send_Distribution is not completely successful. It is responsible for reporting this to the destination agent via a local server report.

Alternatively, the specific server may choose to notify DS of the partially successful operation. It does so by returning an "early acceptance" indication (more precisely, a SPECIFIC_SERVER_EXCEPTION return code) followed by a server report. Upon receipt of this indication, DS continues processing the distribution normally, but terminates the auxiliary server operation. DS delivers the server report to the destination agent on the Receive_Distribution verb.

DS does not terminate the distribution because of a specific-server exception at the destination. Since DS has succeeded in transporting the distribution through the DS network, termination of the distribution because of an application (server) condition is not appropriate. DS continues processing the distribution normally, and reports the specific-server exception to the local agent.

Direct Fetch and Store

Implementations may choose not to perform auxiliary server operations at the origin and destination. At the origin, the copy-making step from the specific server to the general server may be bypassed, so that the server object is not retrieved from the specific server until DS is ready to send the distribution into the network. This implementation elective is referred to as *direct fetch*. Because of the potential reporting complexity, direct fetch is performed only for distributions that do not require fan-out at the origin. If a distribution requires origin fan-out, the origin DSU copies the server object to the general server before sending the distribution.

An implementation elective corresponding to direct fetch may be performed by a destination DSU. The destination DSU may choose to store an incoming object directly into an application's (or user's) private space as the DTMU is being received. This elective is referred to as *direct store*.

The time during which a DTMU is being received at a DSU is referred to as *receive time*. The corresponding time at the sending DSU is referred to as *send time*. If any type of direct storing is to be performed, the command portion of the DTMU must be analyzed at receive time to determine the destination server name. If direct storing by a nonreversible server is to be performed, the destination list must also be analyzed at receive time to determine whether or not all the destinations are local. This implies that both the routing and directing functions are at least partially performed while the DTMU is being received. For some DSUs, such an analysis might be difficult. For others, such as a DSU with only one user, it might be trivial.

If the destination server name is not supported by the DSU or, for nonreversible servers, if some of the destination users turn out not to be local, then direct storing is unachievable and the general server is invoked at receive time.

Exception Handling with Direct Fetch and Store

If direct fetch or direct store is used in place of an auxiliary operation, DS reports exceptions based on the same rules that apply to auxiliary operations. Specific-server exceptions are reported via local server reports; DS exceptions result in distribution reports.

At the origin, an exception detected by the specific server results in a local server report; the distribution is aborted (it could not be accepted by DS, since the server object could not be obtained). An exception detected by DS results in a distribution report.

At the destination, an exception detected by DS results in the termination of the distribution and the generation of a distribution report. An exception detected by the specific server results in a local server report; the distribution is not terminated, but is delivered to the destination agent.

The destination server returns an "early acceptance" indication (i.e., a SPECIFIC_SERVER_EXCEPTION return code) during direct store to inform DS that an exception has occurred and that the server does not wish to receive the remainder of the server object. The receiving DSU may then choose whether or not to inform the sending DSU. If the receiving DSU does not inform the sending DSU, the receiving DSU discards the server object as it is received rather than writing it to the server. Alternatively, the receiver may inform the sender via a Receiver Exception Message Unit. The two DSUs then use mid-MU restart protocols to continue the transfer of the remainder of the DTMU. If the remainder of the DTMU is transferred successfully, the destination DSU passes the server report to the destination agent as part of the Receive_Distribution verb.

Server Access Descriptors and Specific Server Information

If a specific-server object is to be included in the distribution, the originating agent includes the specific server name in the distribution request. The agent also includes one of two additional parameters: *server_access* or *specific_server_info*. The *server_access* parameter is used to specify a "pointer" to the server object. The *specific_server_info* parameter is used to specify instructions for the specific server to use to process the object prior to presenting it to DS.

DS uses *server_access* or *specific_server_info* in much the same way, presenting the parameter to the specific server on the Initiate_Read verb. The server uses the parameter to return a stream of bytes (the object) to DS. If one or more of the destinations of the distribution are local to the origin DSU, however, DS may treat the server object differently depending on whether *server_access* or *specific_server_info* is supplied.

The *server_access* parameter implies that the operation performed by the origin specific server is one of simple retrieval. If the distribution contains destinations local to the origin DSU, DS may choose to bypass the specific server operation for those destinations, and merely supply the *server_access* parameter to the destination agent on the Receive_Distribution verb. No specific server operation would be performed to store the specific server object on behalf of the local destinations.

The *specific_server_info* parameter implies that the specific server performs some sort of application-specific processing of the object prior to presenting it to DS. If this parameter is specified, DS does not bypass the specific server operation for local destinations. DS invokes the specific server to "read" the object and perform the application-specific processing indicated by *specific_server_info.* (In some cases, the *specific_server_info* might contain all the data returned on the Read verb.) DS then invokes the specific server again to store the object for the local destinations.

DS issues access-management verbs to control access to specific server objects described by *server_access.* No access-management verbs are used when *specific_server_info* is supplied in the distribution request.

Agent vs. Server Objects

Small amounts of data that the origin agent wishes to transfer directly to the destination agent are encoded in the agent object. Although the destination server is allowed read-only access to this data, the agent object is not delivered directly to the destination server nor to general servers at intermediate DSUs. The server object is intended for larger objects, or objects that require application-specific processing by servers. The origin application program may use both the agent and server objects to achieve a "double-barreled" flow to the destination agent and server: the origin agent sends information to the destination agent in the agent object; the origin server sends information to the destination server in the server object. This is illustrated in Figure 27 on page 50.

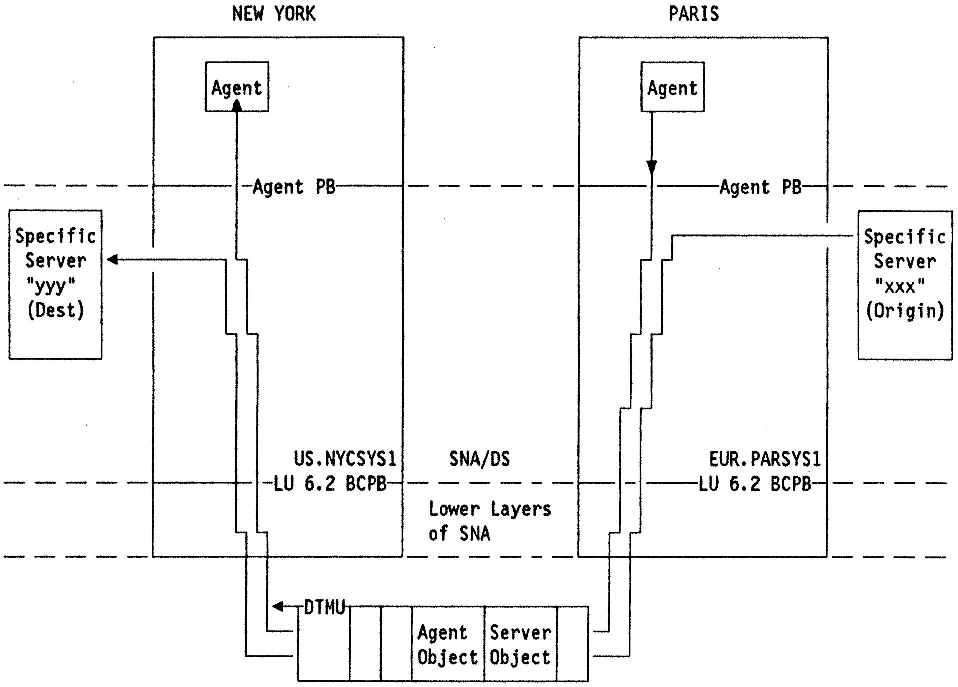


Figure 27. Flow of Agent and Server Objects

The Server Protocol Boundary

Servers are outside the DS architecture. Between the servers (there could be several) and the DS processes is a protocol boundary much like the agent protocol boundary. Instead of requests and deliveries, the server protocol boundary has server objects and certain control information passed across it.

The verbs that define the functions of passing the server object across the server protocol boundary are:

- Initiate_Read
- Read
- Terminate_Read
- Initiate_Write
- Write
- Terminate_Write

If an exception occurs after an object has been completely stored (and Terminate_Write has been issued), the DSU may have to instruct the server to delete the object previously stored and back out any application-specific processing performed during the storing operation. The DSU issues this instruction via the verb

- Backout_Server_Object

Before DS accepts responsibility for distributing an object, it establishes "access rights" to prevent any rewriting or deletion of the object before DS is finished sending it out. Two access verbs provide this capability:

- Assign_Read_Access
- Release_Read_Access

These access verbs are used by DS and the agent (originator or destination) to accomplish the transfer of responsibility for the server object, and to ensure that the object is maintained by the server while the agent or DS requires access to it.

Once a server object has been created and the first read-access rights have been assigned, the object cannot be modified or deleted until all read-access rights have been released. The general server automatically deletes a server object in its space when the object's read-access list becomes empty. The deletion action for specific server objects depends upon the specific server.

DS uses two additional verbs to exchange information with the server regarding mid-MU restart of the server object:

- Query_Last_Byte_Received
- Terminate_Restartability

DS implementations capable of byte-count restart (see "Mid-MU Restart" on page 66) specify a restartability parameter on the Initiate_Read and Initiate_Write verbs to request that the server maintain a count of the number of bytes processed. The DSU queries the restart information by using the Query_Last_Byte_Received verb. When the restart information is no longer needed, the DSU issues Terminate_Restartability to allow the server to discard it.

DS and LU 6.2

The Distribution Transport Sublayer

The DS distribution transport sublayer consists of two transaction programs: DS_Send and DS_Receive. These transaction programs issue LU 6.2 basic conversation verbs to control the sending and receiving of DMUs over LU 6.2 conversations. DS_Send is specialized to act as the sender of DMUs; DS_Receive is specialized to act as the receiver of DMUs.

Each instance of DS_Send or DS_Receive acts as the endpoint of a single LU 6.2 conversation. DS is designed to operate efficiently on a single-session connection, but to exploit parallel sessions if they are available. If the LU at which a DSU resides offers parallel sessions, the DSU may activate multiple instances of DS_Send and DS_Receive to make use of all the available sessions.

The DS transaction program that performs the directing and routing functions is called "DS_Router_Director." As the router-director operates on outbound distributions, it places those distributions on *next-DSU queues*. For the purposes of this discussion, all the distributions that are awaiting transmission to a particular LU using a particular LU 6.2 mode name may be viewed as residing on a single next-DSU queue for that *LU name, mode name* combination (see "Selecting Send Order Using Service Parameters" on page 29 for more about next-DSU queues).

When DS_Router_Director places a distribution on a next-DSU queue, it checks that an instance of DS_Send is available to send the distribution to the partner DSU. Depending on the implementation, the number of sessions available for use by DS, and the number of instances of DS_Send already active, a new instance may or may not be started. Implementations monitor the number of active instances of DS_Send so that a session is available before activating an additional instance.

When an instance of DS_Send is started locally (i.e., rather than via an Attach from DS_Receive) for a particular *LU name, mode name* connection, it allocates a conversation with an instance of DS_Receive at the partner DSU. DS_Send then scans the next-DSU queue serving that LU name and mode name, and retrieves the highest-priority distribution from the queue. It encodes the distribution as a DMU and sends it to DS_Receive. If it is able to successfully send the distribution and if DS_Receive accepts responsibility for it, DS_Send returns to the next-DSU queue, retrieves the next-highest-priority distribution, and continues the process.

DS_Send's algorithm for selecting entries from the next-DSU queue means that distributions of higher priority are serviced before distributions of lower priority, and that distributions of the same priority are serviced in first-in, first-out (FIFO) order. DS_Send continues sending distributions until the next-DSU queue is empty or until an exception causes the conversation to be deallocated.

If multiple instances of DS_Send are active for the same *LU name, mode name* combination, each one returns to the next-DSU queue independently to retrieve and ship the next waiting distribution. Figure 28 on page 53 illustrates this parallel-session scenario, with several instances of DS_Send serving the next-DSU queue for a connection.

DS traffic flow is normally "push oriented." In other words, the flow of DMUs is normally initiated by DS_Send, when new traffic arrives at a DSU. A new instance of DS_Send is started, if necessary; DS_Send then allocates a conversation and begins sending DMUs.

It is also possible, however, for DS_Receive to "pull" traffic from a partner DS_Send. To do so, DS_Receive simply allocates a conversation to DS_Send and enters receive state. When DS_Send is placed in send state, it begins sending any traffic on its next-DSU queues. DS_Receive is responsible for soliciting traffic in this manner after it has reported a recoverable exception to DS_Send. The recoverable exception causes DS_Send to stop sending traffic; DS_Receive starts the traffic flow again when the exception condition has been remedied. DS_Receive might also choose to solicit traffic in other situations, such as when a switched connection is being used.

DS_Send and DS_Receive issue LU 6.2 basic conversation verbs according to precise protocols defined by the DS architecture. The protocols differ for Format Set 1 and Format Set 2 implementations.

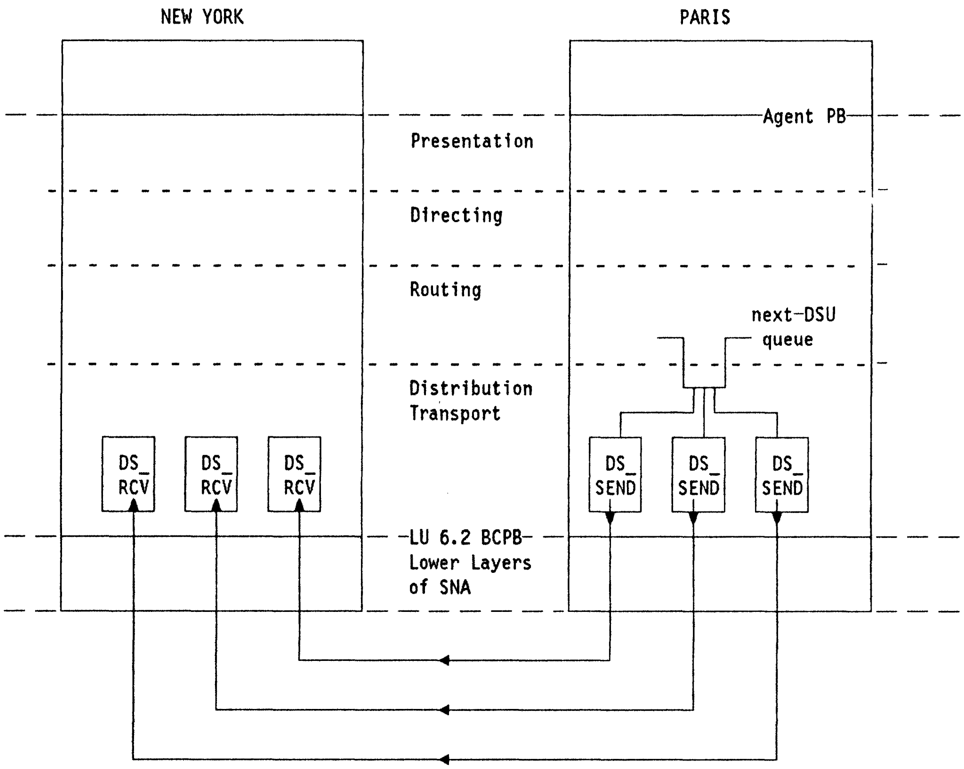


Figure 28. Multiple Instances of DS_Send and DS_Receive

DS's Use of LU 6.2 Verbs -- Format Set 2 Implementations

DS Format Set 2 implementations use a three-way flow to transfer responsibility for a high-integrity distribution from DS_Send to DS_Receive. This protocol not only allows DS_Receive to confirm receipt of the DMU, but also allows the two cooperating DSUs to prevent the generation of duplicate transmissions in cases of network or processor failure. In simplified terms, the protocol works as follows. DS_Send transmits the distribution to DS_Receive, and DS_Receive returns a report confirming that it was received successfully. This report indicates DS_Receive's acceptance of responsibility for the distribution. DS_Send then discards its copy of the distribution, and sends a notification to DS_Receive that the distribution has been discarded. DS_Send may then retrieve and send another distribution, or it may deallocate the conversation.

DS_Send transmits the DMU to DS_Receive by means of one or more LU 6.2 Send_Data verbs. The number of Send_Data verbs required depends on the size of the DMU and the size of the implementation's buffers. DS_Receive issues Receive_And_Wait verbs to receive the DMU. When DS_Send has sent the last part of the DMU, it issues a Receive_And_Wait verb to place DS_Receive in send state. DS_Receive then issues Send_Data to send a Completion Report MU to DS_Send. The Completion Report indicates that DS_Receive has accepted responsibility for the distribution. DS_Receive then returns to receive state by issuing Receive_And_Wait. DS_Send issues Send_Data to transmit a Purge Report MU, confirming that it has discarded its copy of the distribution, and may then begin sending the next distribution. When there are no more distributions to send, DS_Send issues Deallocate Type(FLUSH).

The DS protocol allows for a wide range of throughput. Implementations may send varying amounts of traffic per conversation, depending on choices made by the sender and receiver. The protocol used varies slightly from that described above, depending on those choices.

Levels of Integrity for Distributions

Originators may request one of two levels of integrity for distributions--*basic* or *high*. The three-way flow is used for distributions that specify high integrity. For this type of distribution, DS_Send assigns a number to the DMU before beginning to send it to DS_Receive. This number, known as the MU_ID, is unique for the *LU name, mode name* combination on which the DMU is being sent. It is used for recovery of aborted transmissions and for the prevention of duplicate transmissions in case of network failures. A new MU_ID is used for each connection on which the distribution is sent.

If the distribution specifies basic integrity, DS_Send does not assign an MU_ID to the DMU before beginning transmission, and DS_Send and DS_Receive do not exchange the Completion Report MU and the Purge Report MU. DS_Send transmits the DMU on a "best-effort" basis, and the handling of lost messages or duplicates is left to the using application (the agents).

DS's Use of LU 6.2 Verbs -- Format Set 1 Implementations

Format Set 1 implementations use a protocol similar to that described above. However, instead of confirming the transmission of a DMU with the three-way flow, they use the LU 6.2 Confirm/Confirmed (two-way) flow. That is, after issuing the Send_Data verbs to transmit the DMU, DS_Send issues Confirm. If DS_Receive accepts the DMU, it replies with Confirmed. DS_Send then proceeds with the next DMU.

Format Set 1 implementations do not use the MU_ID for distributions. They use the Confirm/Confirmed protocol for all DMUs.

The use of Confirm/Confirmed normally provides a reliable transfer of responsibility for the DMU from DS_Send to DS_Receive. Some resource failure cases, however, may result in the generation of duplicate distributions. The use of the MU_ID and control MUs by Format Set 2 implementations allows these implementations to detect and prevent duplicate transmissions.

A complete discussion of DS's use of LU 6.2 basic conversation verbs is given in Chapter 2.

Agent Protocol Boundary Verbs

The agent protocol boundary allows use of two types of verbs: those for sending and receiving distributions and those for controlling operations. This section discusses the verbs used by agents to send and receive distributions. The verbs used by agents and operators to control traffic and perform network definition are discussed in "Operations" on page 68.

Verb Overview--Originating Distributions

Agents originate a distribution by issuing a sequence of one or more of the following verbs:

- Send_Distribution initiates a distribution. It is issued either by an origin agent directly on behalf of a user, or by an origin agent performing higher-level function for the DSU. All sending sequences initiated by the origin agent include a Send_Distribution.
- Query_Distribution_Sending is issued by a sending agent to determine the current state of a distribution. In particular, the information returned on this verb informs the agent whether the sending DSU has completed any necessary auxiliary server operations.
- Sending_Sequence_Completed is issued by a sending agent to complete the sending sequence for a high-integrity distribution.

Verb Overview--Receiving Distributions and Reports

Agents use the following verbs to receive distributions and reports from DS:

- Receive_Distribution is issued by the agent to receive a distribution. The distribution is returned to the agent in response to the Receive_Distribution verb.
- Receive_Distribution_Report is issued by an agent to receive a distribution report.
- Receiving_Sequence_Completed is issued after the receiving agent has completed any application-specific processing necessary to store the received distribution.
- Obtain_Local_Server_Report is issued by an agent to obtain a report generated by a local specific server.

Sending Sequences

The sequence of verbs that an agent issues to originate a distribution is called a *sending sequence*. The verbs and parameters included in a sending sequence vary depending on the level of integrity requested for the distribution and the type of server object involved (if any).

The originating agent initiates a sending sequence by issuing Send_Distribution. On the Send_Distribution verb, the agent includes all the control information for the distribution. DS processes the Send_Distribution verb and returns control to the agent.

The remainder of the sending sequence varies depending on the level of integrity requested for the distribution and the type of server object involved. If no specific-server object is involved (e.g., the distribution contained a general-server object or no server object at all), DS is able to take responsibility for the distribution immediately. It indicates this by returning *sending_state* COMMITTED on the Send_Distribution verb. If a specific-server object is involved, DS returns *sending_state* SPEC_SERVER_PENDING instead, because DS cannot "commit" itself to delivering the distribution until the specific-server operation to read in the server object is completed. The *sending_state* for the distribution is changed to COMMITTED after the server object has been successfully retrieved from the specific server.

If the *sending_state* returned on Send_Distribution is SPEC_SERVER_PENDING, the agent must query DS to find out when *sending_state* has been changed to COMMITTED. It does so by issuing Query_Distribution_Sending. The sending state is returned to the agent in response to Query_Distribution_Sending. For a high-integrity distribution, the originating agent is expected to finish the sending sequence by issuing Sending_Sequence_Completed after DS has returned *sending_state* COMMITTED in response to either Send_Distribution or Query_Distribution_Sending. Sending_Sequence_Completed completes the high-integritv transfer of the distribution from the agent to DS. When the agent issues Sending_Sequence_Completed, DS changes *sending_state* to COMPLETED, indicating that DS has responsibility for the distribution and that the sending sequence is complete.

For a basic-integrity distribution, the agent does not issue Sending_Sequence_Completed. When the specific server operation has been completed, the *sending_state* for the distribution is changed from SPEC_SERVER_PENDING to COMPLETED. The agent may issue Query_Distribution_Sending to inquire about the state of the distribution.

The agent supplies the distribution identification and includes it on each verb of the sending sequence. The distribution identification (origin DSU; origin user, if any; origin agent; date; and sequence number) uniquely identifies the distribution.

Sample Sending Sequences

This section presents sample sending sequences for basic- and high-integrity distributions with various types of server objects.

Basic-Integrity Distribution with No Server Object

1. The agent issues Send_Distribution specifying
   - *distribution_identification*
   - *integrity* BASIC
2. DS returns from Send_Distribution immediately with
   - *return_code* OK
   - *sending_state* COMPLETED

Since the distribution specifies basic integrity, the agent does not issue Sending_Sequence_Completed. Responsibility for the distribution has been transferred to DS.

Basic-Integrity Distribution with General-Server Object

1. The agent writes the object into the general server's space using the Initiate_Write, Write, and Terminate_Write verbs.
2. The agent allows DS to have access to the server object by issuing Assign_Read_Access (DS). Eventually the agent will issue Release_Read_Access (*agent_name*) to allow the general server to free the storage space.
3. The agent issues Send_Distribution specifying
   - *distribution_identification*
   - *server* GENERAL
   - *server_access* (*access_descriptor*)
   - *integrity* BASIC
4. DS prevents the server object from being prematurely deleted by issuing Assign_Read_Access (DS).
5. DS returns from Send_Distribution with
   - *return_code* OK
   - *sending_state* COMPLETED

   Responsibility for the distribution has been transferred to DS. The agent does not issue Sending_Sequence_Completed.
6. The agent issues Release_Read_Access (DS), since it can assume DS issued Assign_Read_Access for itself before returning control from Send_Distribution.
7. After sending the distribution, DS issues Release_Read_Access (DS).

Basic-Integrity Distribution with Specific-Server Object

1. The agent stores the object using the specific server, and makes the object available to DS by issuing Assign_Read_Access (DS) to the specific server.
2. The agent issues Send_Distribution specifying
   - *distribution_identification*
   - *server* (*specific_server_name*)
   - *server_access* (*access_descriptor*)
   - *integrity* BASIC
3. DS prevents the server object from being prematurely deleted by issuing Assign_Read_Access (DS) to the specific server.
4. DS returns from Send_Distribution with
   - *return_code* OK
   - *sending_state* SPEC_SERVER_PENDING
5. DS performs an auxiliary operation to copy the server object from the specific server to the general server. After acquiring access to the object in the general server, DS issues Release_Read_Access (DS) to the specific

server. When the copy is completed, DS changes the sending state to COMPLETED.

Alternatively, DS may perform direct fetch, deferring its invocation of the specific server until it sends the DTMU. The sending state is not changed to COMPLETED until after the object has been read from the specific server.

6. The agent does not issue Sending_Sequence_Completed. It may issue Query_Distribution_Sending to inquire about the state of the distribution.
7. The agent issues Release_Read_Access (DS) to the specific server.
8. DS issues Release_Read_Access (DS) to the appropriate server after the distribution has been sent.

Basic-Integrity Distribution with Specific-Server Information

1. The agent, by application-specific means, makes the server object (if any) available when needed. No access verbs are issued when specific server information is supplied instead of an explicit access descriptor (i.e., the *server_access* parameter) for a server object.
2. The agent issues Send_Distribution with
   - *distribution_identification*
   - *server* (*specific_server_name*)
   - *specific_server_info* (*information*)
   - *integrity* BASIC
3. DS returns control from Send_Distribution immediately with
   - *return_code* OK
   - *sending_state* SPEC_SERVER_PENDING
4. DS performs an auxiliary server operation to read the specific server object from the specific server and write it to the general server. When the auxiliary operation has completed, DS changes the sending state to COMPLETED. DS issues access management verbs to the general server, but not to the specific server. If DS performs direct fetch instead of an auxiliary operation, the sending state is not changed to COMPLETED until the direct fetch is completed.
5. The agent does not issue Sending_Sequence_Completed for the basic integrity distribution. The agent may issue Query_Distribution_Sending to inquire about the state of the distribution.

High-Integrity Distribution with No Server Object

1. The agent issues Send_Distribution specifying
   - *distribution_identification*
   - *integrity* HIGH
2. DS returns from Send_Distribution immediately with
   - *return_code* OK
   - *sending_state* COMMITTED
3. The agent completes the transfer of responsibility for the distribution by issuing Sending_Sequence_Completed.

4. DS returns from Sending_Sequence_Completed with

   - *return_code* OK
   - *sending_state* COMPLETED

High-Integrity Distribution with General-Server Object

1. The agent writes the object into the general server's space using the Initiate_Write, Write, and Terminate_Write verbs.

2. The agent allows DS to have access to the server object by issuing Assign_Read_Access (DS). Eventually, the agent will issue Release_Read_Access (*agent_name*) to allow the general server to free the storage space.

3. The agent issues Send_Distribution specifying

   - *distribution_identification*
   - *server* GENERAL
   - *server_access* (*access_descriptor*)
   - *integrity* HIGH

4. DS prevents the server object from being prematurely deleted by issuing Assign_Read_Access (DS).

5. DS returns from Send_Distribution with

   - *return_code* OK
   - *sending_state* COMMITTED

6. The agent completes the transfer of responsibility for the distribution by issuing Sending_Sequence_Completed.

7. DS returns from Sending_Sequence_Completed with

   - *return_code* OK
   - *sending_state* COMPLETED

8. The agent issues Release_Read_Access (DS), since DS has issued Assign_Read_Access for itself.

9. After sending the distribution, DS issues Release_Read_Access (DS).

High-Integrity Distribution with Specific-Server Object

1. The agent stores the object using the specific server, and makes the object available to DS by issuing Assign_Read_Access (DS) to the specific server.

2. The agent issues Send_Distribution specifying

   - *distribution_identification*
   - *server* (*specific_server_name*)
   - *server_access* (*access_descriptor*)
   - *integrity* HIGH

3. DS prevents the server object from being prematurely deleted by issuing Assign_Read_Access (DS) to the specific server.

4. DS returns from Send_Distribution with

   - *return_code* OK
   - *sending_state* SPEC_SERVER_PENDING

5. DS performs an auxiliary operation to copy the server object from the specific server to the general server. After acquiring access to the object in the general server, DS issues Release_Read_Access (DS) to the specific server. When the copy is completed, DS changes the sending state to COMMITTED.

   Alternatively, DS may perform direct fetch, deferring its invocation of the specific server until it sends the DTMU. The sending state is not changed to COMMITTED until after the object has been read from the specific server.

6. The agent determines the state of the distribution by issuing Query_Distribution_Sending, specifying the *distribution_identification*. The agent may have to issue Query_Distribution_Sending more than once, until *sending_state* is returned as COMMITTED.

7. When DS returns *sending_state* COMMITTED in response to Query_Distribution_Sending, the agent completes the high-integrity transfer of responsibility by issuing Sending_Sequence_Completed.

8. DS returns from Sending_Sequence_Completed with

   - *return_code* OK
   - *sending_state* COMPLETED

9. The agent issues Release_Read_Access (DS) to the specific server.

10. DS issues Release_Read_Access (DS) to the appropriate server after the distribution has been sent.

High-Integrity Distribution with Specific-Server Information

1. The agent, by application-specific means, makes the server object (if any) available when needed. No access verbs are issued when specific server information is supplied instead of an explicit access descriptor for a server object.

2. The agent issues Send_Distribution with

   - *distribution_identification*
   - *server* (*specific_server_name*)
   - *specific_server_info* (*information*)
   - *integrity* HIGH

3. DS returns control from Send_Distribution immediately with

   - *return_code* OK
   - *sending_state* SPEC_SERVER_PENDING

4. DS performs an auxiliary server operation to read the specific server object from the specific server and write it to the general server. When the auxiliary operation has completed, DS changes the sending state to COMMITTED. DS issues access management verbs to the general server, but not to the specific server. If DS performs direct fetch instead of an auxiliary operation, the sending state is not changed to COMMITTED until the direct fetch is completed.

5. The agent determines the state of the distribution by issuing Query_Distribution_Sending, specifying the *distribution_identification*. The

agent may have to issue Query_Distribution_Sending more than once, until *sending_state* is returned as COMMITTED.

6. When DS returns *sending_state* COMMITTED in response to Query_Distribution_Sending, the agent completes the high-integrity transfer of responsibility by issuing Sending_Sequence_Completed.

7. DS returns from Sending_Sequence_Completed with

   - *return_code* OK
   - *sending_state* COMPLETED

Receiving Sequences

The sequence of verbs that an agent issues to receive a distribution is called a *receiving sequence*. The agent initiates a receiving sequence by issuing Receive_Distribution. DS passes the distribution to the agent in response to the Receive_Distribution verb. After performing any necessary handling and storage of the distribution, the agent issues Receiving_Sequence_Completed. The Receiving_Sequence_Completed verb completes the transfer of responsibility for the distribution from DS to the agent.

Distribution reports are received using a similar receiving sequence. The agent issues Receive_Distribution_Report to obtain the distribution report. After storing the report, the agent issues Receiving_Sequence_Completed to complete the transfer of responsibility.

Sample Receiving Sequences

The two-verb receiving sequence described above is used for both basic- and high-integrity distributions. Sample sequences are given below for the various types of server objects that may be involved in a distribution. Distribution reports are received using a sequence similar to that for a distribution with no server object.

Distribution with No Server Object

1. The agent issues Receive_Distribution. The agent specifies the identifier of the local queue from which the distribution is to be received. The agent may also specify the *distribution_identification* of a particular distribution to be received. If no *distribution_identification* is specified, the first entry in the queue is returned.

2. DS returns from the Receive_Distribution with the control information (i.e., the DS command and the destination list) for the distribution and the agent object (if present).

3. After performing the appropriate agent-specific actions for handling and storing the distribution, the agent issues Receiving_Sequence_Completed. Responsibility for the distribution has been transferred to the receiving agent.

Distribution with General-Server Object

1. The agent issues Receive_Distribution specifying

   - *queue_identifier*
   - (optionally) *distribution_identification*

2. DS returns from Receive_Distribution with the control information for the distribution (including the server access information).

3. The agent issues Assign_Read_Access (*agent_name*) so that the object is not deleted prematurely. DS has previously issued Assign_Read_Access (*agent_name*) to allow the agent to access the object.

4. The agent issues Receiving_Sequence_Completed to complete the transfer of responsibility.

5. After the last copy of the distribution has been accepted by the agent (there could be multiple copies if destination fan-out has been performed), DS issues Release_Read_Access (*agent_name*), since DS can assume that the agent acquired access for itself before issuing Receiving_Sequence_Completed. DS issues Release_Read_Access (DS) to notify the server that DS no longer requires access to the object.

Distribution with Specific-Server Object

1. The agent issues Receive_Distribution specifying

   - *queue_identifier*
   - (optionally) *distribution_identification*

2. DS returns from Receive_Distribution with the control information for the distribution (including the server and access control information). DS has previously stored the object using the specific server.

3. The agent issues Assign_Read_Access (*agent_name*) to acquire access to the object. DS has previously issued Assign_Read_Access (*agent_name*) to allow the agent to access the object.

4. The agent issues Receiving_Sequence_Completed to complete the transfer of responsibility.

5. After the last copy of the distribution has been accepted by the agent (there could be multiple copies if destination fan-out has been performed), DS issues Release_Read_Access (*agent_name*), since the agent has acquired access for itself. DS issues Release_Read_Access (DS) to notify the server that DS no longer requires access to the object.

Distribution with Specific-Server Information

1. The agent issues Receive_Distribution specifying

   - *queue_identifier*
   - (optionally) *distribution_identification*

2. DS returns from Receive_Distribution with the control information for the distribution (including *specific_server_info*). The specific server has previously returned *specific_server_info* to DS in response to the Terminate_Write verb.

3. Access management verbs between the server and DS are not used when *specific_server_info* is returned instead of *server_access*.
4. The agent issues Receiving_Sequence_Completed to complete the transfer of responsibility.

Exception Occurrences and Conditions

In DS, exception occurrences include both conditions detected by various entities at a DSU and intervention by operators. Most types of exceptions consistently produce particular observable conditions.

Some types of exceptions have results that depend on timing conditions or concurrency of events. Such results are difficult to reproduce or even to detect. An example of a timing-dependent exception condition is the generation of duplicate distributions, which is possible in Format Set 1 implementations (Format Set 2 protocols allow implementations to prevent duplicate generation).

DS exception conditions can be classified according to the layer in which they are detected or caused:

- Application--agents and servers
- DS--the DSU itself
- LU 6.2

Application Exceptions: Because of DS's interaction with the application, DS is often aware of exception conditions that are detected or caused at the application layer. For example, a specific-server exception is considered, from the DS perspective, to be an exception at the application layer. If a server exception occurs while DS has responsibility for the distribution, DS may have to terminate the distribution (if told to do so by the server) and, perhaps, report the exception.

DS Exceptions: Exception conditions detected by the DS layer fall into two categories:

- Exceptions detected in the verbs at the protocol boundaries
- Exceptions detected, or operator intervention performed, after DS has accepted responsibility for the distribution.

Exceptions are detected when the control information in the distribution cannot be reconciled with the information in a DSU's tables, or when a resource fails. The exceptions that may be detected are

- Routing exception
- Directing exception
- Format exception
- Function not supported
- System exception
- Others

DSUs are required to perform certain minimal checks before accepting a distribution. These checks are performed by DS presentation services before accepting a distribution from an agent across the agent protocol boundary, and by DS_Receive before accepting a distribution from another DSU. Implementations may choose to perform additional checks before accepting responsibility for distributions. The minimal set of required checks is described in Appendix C.

See Appendix E for a complete list of the exceptions that may be detected by DS.

LU 6.2 Exceptions: DS is aware of LU 6.2 exception conditions because DS must be prepared to accept a variety of responses to the LU 6.2 basic conversation verbs it issues.

Exception Analysis

Within the DSU, exception conditions may be detected at any of the four DS sublayers. As discussed above, an exception may originate within the DSU itself, in the application (the agent or server), or in the LU. The actions taken in response to a particular exception depend on the nature of the exception, the detector of the exception, and the scope of the exception.

Detector: Any of four DS components may detect an exception:

- DS presentation services: The exception is detected at the protocol boundary, while DS is processing a request from an agent.
- DS_Router_Director: The exception is detected while DS is performing routing or directing functions.
- DS_Send: The exception is detected during the sending process.
- DS_Receive: The exception is detected during the receiving process.

Responses to exceptions detected by DS presentation services are discussed in Appendix F. Responses to exceptions detected by other DS sublayers are discussed in Appendix E.

Scope: The scope of the exception condition may be any of the following:

- The DSU: The condition affects all distributions at the DSU.
- The connection: The condition affects all distributions being transmitted, or enqueued for transmission, on a certain connection (*LU name, mode name*).
- The distribution copy: The condition affects one entire distribution copy and is unrelated to the destination list.
- Some but not all destinations of the DMU: One or more destinations are directly affected by the exception, and one or more destinations are unaffected.
- All destinations of the DMU: All destinations in the destination list are equally affected. This is different from a scope of the distribution copy only because the exception is per destination; all destinations happen to be affected.

- Local destinations only: The condition affects only destinations that are local to the DSU.
- To be determined: A temporary condition has occurred that will not affect the distribution (or DSU) unless, after an implementation-defined number of retries has been attempted, the condition is deemed unrecoverable.

As a result of implementation choices concerning role, electives, and optimizations, as described in Appendix C, the exception information generated for a particular condition may vary slightly from one implementation to another. All implementations must be prepared to receive any of the valid report codes, as listed in Appendix E.

Exception Handling

Exception conditions detected by the DS presentation services sublayer are generally reported to the agent as returned parameters on a protocol boundary verb. For exception conditions detected by the routing, directing, or transport sublayers, exception handling procedures may include any of several actions:

- Hold or release the next-DSU queue for the connection, so that transmission to the adjacent DSU is stopped or started. The hold condition placed on the next-DSU queue in response to an exception is termed an *exception-hold*. An exception-hold is released by operator action or by the initiation of a new instance of DS_Send. This contrasts with an *operator-hold*, which is placed on a distribution or a queue by an operator and is released only by operator action.
- Perform MU-level reporting to inform the adjacent DSU, possibly terminating the DMU transmission. For DS_Send, this means sending a Sender Exception MU. For DS_Receive, this means sending a Receiver Exception MU.
- Log the exception condition and notify the operator.
- Abort the distribution, generate a distribution report describing the exception condition, and send it to the report-to DSU or user specified in the distribution.
- Abort the distribution and return a specific-server report to the local agent.
- Purge the distribution from the queue and purge the associated server object, if any.

Any of these actions may be allowed, required, or precluded for a particular exception condition. The tables in Appendix E give the prescribed actions for exceptions detected at various points during the processing of a distribution.

The DSU may perform any of several types of reporting in handling an exception condition. The types of reporting performed by DS are

- Local-agent reporting
- MU-level reporting
- Distribution reporting
- Reporting to the local operator

The types of reporting are described in more detail in Appendix E.

Mid-MU Restart

Conversation failures or resource failures may occur during the transmission of a DTMU. Format Set 2 DS implementations are capable of restarting a failed DTMU at or near the point of failure, rather than having to retransmit the entire DTMU. The term for this capability is *mid-MU restart*.

DS implementations provide two types of mid-MU restart. All Format Set 2 implementations provide the capability to resume the transmission of a failed DTMU at the beginning of any of the highest-level structures following the agent object. Since the highest-level structures of a DTMU are sometimes referred to as LLIDs (because of the format of their encoding) this type of mid-MU restart is referred to as *LLID restart*.

DS implementations may elect to provide an additional type of mid-MU restart called *byte-count restart*. A DSU that provides byte-count restart is capable of restarting a transmission at any byte position within the server object.

To provide mid-MU restart, DSUs maintain knowledge of the high-level structures of the DTMU already received. Implementers of byte-count restart, in conjunction with their servers, also maintain knowledge of the number of bytes of the server object already received. After a failure during the transmission of a DTMU, the receiver notifies the sender of the last structure it received and, optionally, of the last byte within the server object it received. The sender then continues the transmission based on that information.

Further information on the mid-MU restart elective is given in the implementation model in Chapter 3 and in Appendix C.

Distribution Reporting

In DS, two kinds of reports are returned to agents. One kind is the *local server report*, which is generated by a specific server and given to the local agent by DS. Local-server reports inform the agent of an application-specific condition detected by the specific server.

The other kind of report is the *distribution report*. The originator may specify, in the distribution request, that DS is to report on the condition of the distribution. For example, the originator may wish to be informed if DS is unable to deliver the distribution. Then, if an exception occurs, DS generates a report and delivers it to the report-to user (or DSU) named in the request.

Distribution reporting may be requested for high-integrity distributions only (not for basic-integrity distributions).

Distribution Report Message Units

When distribution reports are sent through the network, DS encodes them as distribution report message units (DRMUs). The structure of a DRMU is shown in Figure 29 on page 67. Like a DTMU, a DRMU is introduced by a prefix and concluded by a suffix. The command contains the control information for the DRMU; the report information and SNA Condition Report describe the particular condition being reported. The report-to DSU/user is the DSU or user to which the report is being sent.

Prefix	Command	Report-To DSU/User	Report Information	SNA Condition Report	Suffix

Figure 29. Distribution Report Message Unit Structure

Service Parameters in the DRMU

Like the DTMU, the DRMU carries service parameters that specify handling instructions for the distribution report. The parameters that may be specified in the DRMU are *priority*, *protection*, and *security*. There is no *capacity* parameter in the DRMU because the DRMU does not carry a server object. Any or all of the service parameters may be omitted from the DRMU. Default service levels are defined (see Appendix G) for each service parameter; DSUs processing the DRMU assume the default values for any parameters that are omitted.

The originator of the distribution may specify *report service parameters* explicitly in the distribution request. If report service parameters are specified, they are used as the service parameters in any DRMUs that are generated as part of the distribution.

If the originator does not specify one or more of the report service parameters, a DSU that generates a report derives appropriate service parameters for the DRMU from the service parameters in the DTMU. For the *protection* and *security* parameters, the comparison operator and value derived are the same as those specified (either explicitly or via defaults) in the DTMU. For example, if the DTMU specified *protection* (REQUIRE_LEVEL_GE LEVEL2), the same parameter is used in the DRMU.

For the *priority* service parameter, the value derived is either FAST or CONTROL. FAST is used if the DTMU specified FAST priority; CONTROL is used if the DTMU specified a DATA_N priority. CONTROL priority is used only in DRMUs; it may not be specified for the priority service parameter in a DTMU.

The originator explicitly specifying a report service parameter for *priority* may specify a value of FAST, CONTROL, or DATA_N.

The Report-To Agent

If the originating agent requests reporting on a distribution, it may specify the particular agent that is to be invoked to receive any distribution reports. If no report-to agent is specified, the agent invoked will be the same as the origin agent for the distribution.

A report-to agent might be specified if the origin agent is not appropriate for receiving report information. For example, if a large number of destinations is specified, it might be desirable to have a special registry agent keep track of all the destinations, noting any errors encountered as reports are received. The originator could then get a consolidated report on the entire distribution.

Third-Party Reporting

Besides specifying the destination of the distribution, the originating agent may specify a third party to which all distribution reports are to be sent. This *report-to* party may be a user or a DSU. If no report-to user or DSU is specified, reports are sent to the originating user or DSU. The originator specifying a third party for reporting should be reasonably confident that both the report-to DSU/user and a route to that DSU/user exist.

Operations

The agent protocol boundary consists of two types of verbs. Verbs used to send and receive distributions are discussed in "Agent Protocol Boundary Verbs" on page 55. The architecture also defines operations verbs, which are used by agents and operators to manage DS traffic, manage DS connections, and maintain DSU definitions. The operations verbs are listed below.

Managing Distributions

Operations verbs permit agents to manage the distributions they have originated or that have been queued for delivery to them.

- List_Queues_Containing_Distribution returns the queue identifier of every queue that contains a copy of the specified distribution.
- List_Queue_Entries returns a list of the entries in a specified queue.
- Get_Distribution_Info returns the control information for a specified distribution.
- Purge_Queue_Entry deletes a copy of a given entry from a given queue.
- Hold_Distribution_Copy places a hold on a given copy of a given distribution on a given queue.
- Release_Distribution_Copy releases the hold on a given copy of a given distribution on a given queue.

Managing Connections

Some DS implementations choose to exercise close control over the number and usage of a connection's conversations (and underlying sessions). These verbs for managing connections are limited to operations agents; they are not meant to be issued by ordinary user agents.

- List_Conversations lists the number of active conversations, both sending and receiving, for a connection, or an adjacent DSU, or for all adjacent DSUs.
- List_Distributions_Being_Sent lists the IDs, including size of the yet-to-be sent portion of the server object, for all the distributions currently in the process of being sent on a connection, or on all connections to an adjacent DSU, or on all active connections.
- List_Distributions_Being_Received lists the IDs and attributes, including the size of the already-received portion of the server object, for all distributions currently in the process of being received on a connection, or on all connections to an adjacent DSU, or on all active connections.

- List_Adjacent_DSUs lists selectively the DSUs adjacent to the local DSU.
- List_Connections lists selectively the connections available to the DSU.
- List_Control_MU_Queue lists the entries on a specified Control MU queue.
- Start_Connection causes the appropriate number of DS_Send instances to be started. A parameter of this verb can be used to set the maximum number of instances of DS_Send that the DSU is allowed to start.
- Reset_MU_ID_Registry allows an operator to resynchronize the MU_ID registry for a connection.
- Terminate_Connection causes every conversation on the connection to be stopped, either abruptly, or with a suspend, or as the MU in progress completes.
- Terminate_Conversation causes a particular conversation to be stopped, either abruptly, or with a suspend, or as the MU in progress completes.
- Reroute_Distribution_Copies causes all, or a selection of, the distribution copies found on a next-DSU queue to be reprocessed through the routing logic. Typically, the next-DSU queue will have been placed in the inactive state and the rerouting will queue the distributions for alternate next-DSUs.

Maintaining DSU Definitions

The DS system-definition verbs are used to create and maintain the various tables and miscellaneous values that constitute the definition of a particular DSU. The verbs are described below as they would be issued at the local operations protocol boundary.

The verbs used to support DSU definition are the following:

- List_DSU_Data is used to display to the operator the value of a specified system parameter or the entries in a defined table.
- Add_DSU_Data is used to add an entry containing one or more data parameters to a data structure that can contain a list of entries.
- Remove_DSU_Data is used to remove an entry containing one or more data parameters from a data structure that can contain a list of entries.
- Modify_DSU_Data is used to change the value of one or more data parameters in a data structure. This verb is used for mandatory parameters of the DSU definition, such as the DSU name. It may also be used to modify a portion of an existing list entry.

The DS data-structures that constitute the definition of a DSU and are maintained by the above verbs are the following:

- Directory
- Routing table
- Intervention list
- DSU definition
- Connection definitions
- Next-DSU queue definitions
- Agent list
- Server list

- MU ID registry

Managing Logs

- Get_Exception_Log_Entry returns a given exception logged by DS.
- Get_Distribution_Log_Entry returns the message-unit control information logged by DS.

Chapter 2. Overview of SNA/DS Protocols

The DS Distribution Transport Sublayer

This chapter describes DS′s use of LU 6.2 for Format Set 2 implementations. For a detailed model of DS′s use of LU 6.2, refer to Chapter 3.

A brief introduction to the DS distribution transport sublayer is given in Chapter 1. This sublayer is composed of two transaction programs called DS_Send and DS_Receive. Each instance of DS_Send or DS_Receive acts as the endpoint of a single LU 6.2 conversation. Parallel conversation usage is achieved via multiple instances of the transaction programs.

In general, traffic is sent in both directions between a pair of DSUs. Instances of DS_Send at each DSU transmit distributions to instances of DS_Receive at the partner. The sessions between the DSUs may be grouped according to mode name. Figure 30 shows two DSUs communicating over parallel sessions with two different mode names. Distributions are sent in only one direction on a given conversation, from DS_Send to DS_Receive. Control message units may flow in either direction on the conversation, from DS_Send to DS_Receive or from DS_Receive to DS_Send.

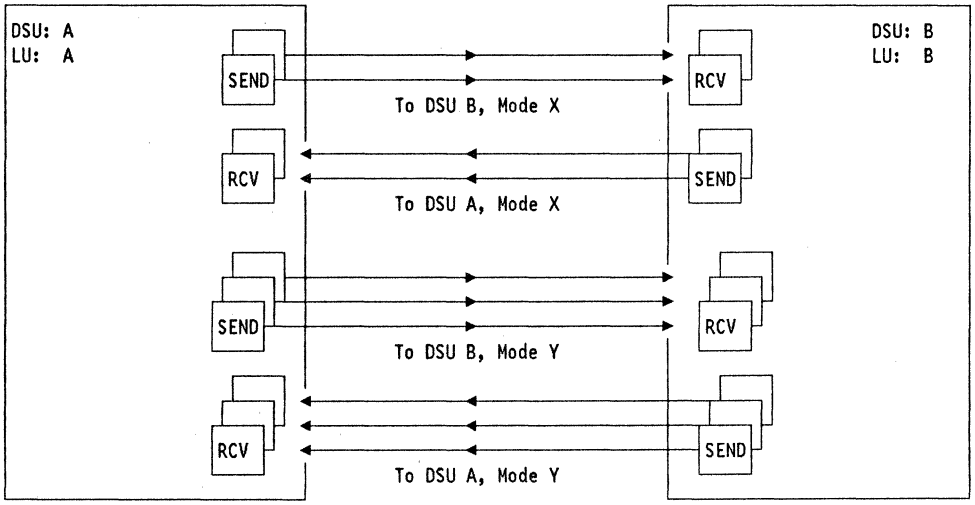


Figure 30. Parallel Session Usage Between Two DSUs

To simplify the following discussion, Figure 31 on page 72 illustrates the main components of the distribution transport sublayer for a single direction of transfer on a single mode name (FAST). At DSU A, three instances of DS_Send are sending traffic on a connection to DSU B. The connection is identified by the combination of the name of the LU at which the partner DSU resides and the mode name of the conversations over which the DSUs communicate. Except for the Router-Director queue, separate instances of the data structures shown in Figure 31 are maintained for each connection (i.e., each *LU name, mode name* pair).

Data Structures at the Sending DSU

All three instances of DS_Send obtain the distributions that they send to their partner from the same next-DSU queue. As discussed in Chapter 1, an implementation might use one or more than one next-DSU queue for each connection. For the purposes of this discussion, assume all distributions awaiting transmission on a particular connection reside on a single queue.

Besides the DMUs they transmit to the partner DSU, the DS_Send instances generate and transmit control MUs, which they use to inform the partner DSU about the status of DMUs. In the DS model in this document, control MUs are placed on a separate queue (the control-MU queue) when they are generated, and are transmitted by the first instance of DS_Send that becomes available. Whenever an instance of DS_Send enters send state, it checks the control-MU queue; if it finds entries there, it transmits them. If the control-MU queue is empty, DS_Send checks the next-DSU queue and transmits the highest-priority distribution it finds there.

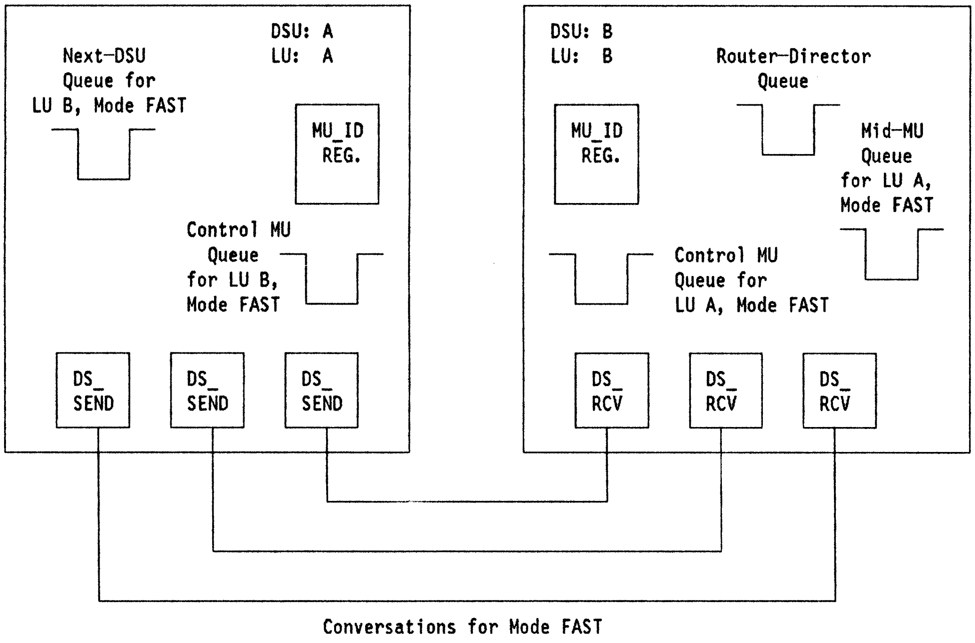


Figure 31. Components of the DS Distribution Transport Sublayer

Each instance of DS_Send transmits MUs over a single LU 6.2 conversation to a partner instance of DS_Receive. DS_Send is specialized to send DMUs; DS_Receive is specialized to receive DMUs. Both DS_Send and DS_Receive send and receive various types of control MUs.

Data Structures at the Receiving DSU

At the partner DSU, instances of DS_Receive (three, in the sample configuration in Figure 31) receive DMUs and control MUs from the corresponding instances of DS_Send. After an instance of DS_Receive successfully receives a DMU, it enqueues a control block representing the DMU on the router-director queue for further processing.

The instances of DS_Receive share a control-MU queue for the connection, from which they transmit control MUs to the instances of DS_Send. Whenever an instance of DS_Receive enters send state, it obtains the first entry on the control MU queue and transmits it to DS_Send. It continues sending control MUs until the control-MU queue is empty, and then returns to receive state.

The instances of DS_Receive also share a mid-MU restart queue for the connection. When DS_Receive begins receiving a DMU, it makes an entry for that DMU on the mid-MU restart queue. As DS_Receive receives the DMU, it updates its control block on the mid-MU restart queue. If an exception occurs and the DSUs decide to attempt mid-MU restart, DS_Receive informs DS_Send of the point at which retransmission should begin based on the information in the mid-MU restart queue entry. When DS_Receive receives the continuation of the DMU, it finds the queue entry and continues updating it. Finally, when the DMU has been completely received, DS_Receive enqueues its control block on the router-director queue and removes it from the mid-MU restart queue.

The MU_ID Registries

Each DSU shown in Figure 31 on page 72 contains an MU_ID registry for the connection. The MU_ID registry tracks the state of each MU_ID currently in use by the DSU. An MU_ID is a number representing a particular piece of work in progress between the transport sublayers of the two DSUs.

MU_IDs are used only when transmitting distributions that specify high integrity. Since no confirmation flows are exchanged for basic-integrity distributions, MU_IDs are not assigned to them. Basic-integrity distributions are transmitted on a best-effort basis; exceptions that occur during transmission may cause a basic-integrity distribution to be lost.

If a distribution on the next-DSU queue specifies high integrity, it is assigned an MU_ID by the sender when it is first selected for transmission to the partner DSU. The MU_ID uniquely identifies the distribution until the two DSUs agree that they have finished (successfully or unsuccessfully) the piece of work represented by that MU_ID. At any time during the transmission of the distribution, the MU_ID is in a particular state; when the DSUs finish using it, it is marked PURGED by both DSUs. Each DSU updates the state of the MU_ID as it processes the distribution, and as it receives control MUs containing information about the MU_ID from the partner. Refer to "Management of Message Unit IDs" on page 81 for more information about the use of MU_IDs and their states.

Figure 31 on page 72 illustrates the transport sublayer for a single connection (*LU name, mode name* combination). Except for the router-director queue (of which there is typically only one per DSU), each DSU contains an instance of each data structure shown in Figure 31 for each connection (*LU name, mode name* pair) over which it communicates.

Defining Connections

A DS connection is defined as the set of LU 6.2 sessions using a particular mode name over which a DSU communicates with another DSU. A connection is identified by both the *LU name* of the partner DSU and the *mode name* of the LU 6.2 sessions used to communicate with the partner. A connection may consist of one or more than one session. Multiple connections using different *mode names* may exist between two DSUs.

LU 6.2 control operator verbs allow an operator to define to the LU the maximum number of sessions allowed for a given connection (by specifying the LU name and mode name), the contention-winner polarity of the sessions, and the number of sessions that are to be automatically activated by the LU. The DS operations verb, Start_Connection, allows the operator to define to the DSU the maximum number of instances of DS_Send that may be active on the connection. Since each active instance of DS_Send uses one session, and since typically each DSU uses its contention-winner sessions for sending, the value of the *max_DS_Sends* parameter should normally be less than or equal to the number of contention-winner sessions defined for the DSU on the connection.

DS Protocol for Transmitting Distributions

DS_Send and DS_Receive issue LU 6.2 basic conversation verbs to transmit message units between DSUs. The message units they transmit are of two general types: distribution message units (DMUs) and control message units (CMUs). Distribution message units contain distributions (or portions of distributions); control message units contain information about DMUs and conversation control information.

The information contained in DMUs typically travels through more than one DSU as it makes its way through the network. The information contained in CMUs, however, is used strictly between adjacent DSUs to manage the traffic between those DSUs.

The various types of DMUs sent from DS_Send to DS_Receive are:

- **Distribution-Transport MU (DTMU):** DS uses DTMUs to move distributions through the DS network.
- **Distribution-Report MU (DRMU):** DS uses DRMUs to send distribution reports.
- **Distribution-Continuation MU (DCMU):** DS uses DCMUs to resume transmission of suspended DTMUs.

The various types of CMUs exchanged between DS_Send and DS_Receive are listed below:

- **Sender-Exception MU (SEMU):** The SEMU is sent from DS_Send to DS_Receive to inform DS_Receive that DS_Send has encountered an exception condition.
- **Receiver-Exception MU (REMU):** The REMU is sent from DS_Receive to DS_Send to inform DS_Send that DS_Receive has encountered an exception condition.

- **Completion-Query MU (CQMU):** The CQMU is sent from DS_Send to DS_Receive to inquire about the state of an MU_ID.
- **Completion-Report MU (CRMU):** The CRMU is sent from DS_Receive to DS_Send to inform DS_Send of the state of an MU_ID or to control traffic flow on a conversation.
- **Purge-Report MU (PRMU):** The PRMU is sent from DS_Send to DS_Receive to inform DS_Receive that the piece of work represented by a particular MU_ID is ended, and that the MU_ID has been marked PURGED.
- **Reset-Request MU (RRMU):** The RRMU is sent from DS_Send to DS_Receive to request that DS_Receive reinitialize its MU_ID registry.
- **Reset-Accepted MU (RAMU):** The RAMU is sent from DS_Receive to DS_Send after DS_Receive has reinitialized its MU_ID registry in response to an RRMU.

Each of these control MUs will be discussed in detail in subsequent sections.

Integrity of Distributions

DS uses a three-step flow to transfer high-integrity DMUs from DS_Send to DS_Receive. Figure 32 on page 76 gives a conceptual description of the messages sent between the DSUs to accomplish a single transfer.

After transmitting the DMU, the sender issues a "Responsibility-Transfer Requested" message. In the DS protocol, the suffix of the DMU carries the "Responsibility-Transfer Requested" semantics. By sending the suffix, DS_Send informs DS_Receive that it has sent the entire DMU, and requests that DS_Receive accept responsibility for it.

Once the sender has issued "Responsibility-Transfer Requested," it awaits notification as to whether the receiver has accepted or rejected the DMU. The sender takes no action (such as rerouting the distribution or sending a distribution report) that might result in a duplicate DMU until it has been informed that the receiver has not accepted the DMU.

If DS_Receive is able to complete the processing for the DMU, it returns a "Responsibility Accepted" message. "Responsibility-Accepted" is denoted in DS by the Completion-Report MU specifying that the MU_ID is COMPLETED. The CRMU (COMPLETED) notifies DS_Send to delete its copy of the distribution, since DS_Receive has safely received it.

After deleting its copy of the distribution, DS_Send issues a "Responsibility-Relinquished" message to DS_Receive. DS uses the Purge-Report MU to convey the "Responsibility-Relinquished" message. The PRMU informs DS_Receive that it need no longer retain explicit memory of the state of the MU_ID, because the unit of work represented by the MU_ID is finished.

The three-step flow allows implementations to prevent the loss or duplication of distributions, because any of the three messages may be re-sent at any time. If the original transmission is lost, DS_Send may retry it (after querying DS_Receive to check that it has not been received). If the CRMU (COMPLETED) is lost, DS_Receive may send another (perhaps in response to a query from

DS_Send). If the PRMU is lost, DS_Receive may re-issue the CRMU (COMPLETED) to solicit another PRMU. Each DSU retains the information required to re-send its messages until it is sure that the partner has received them.

The three-step flow is used only for high-integrity distributions. No confirmation flows are exchanged for basic-integrity traffic.

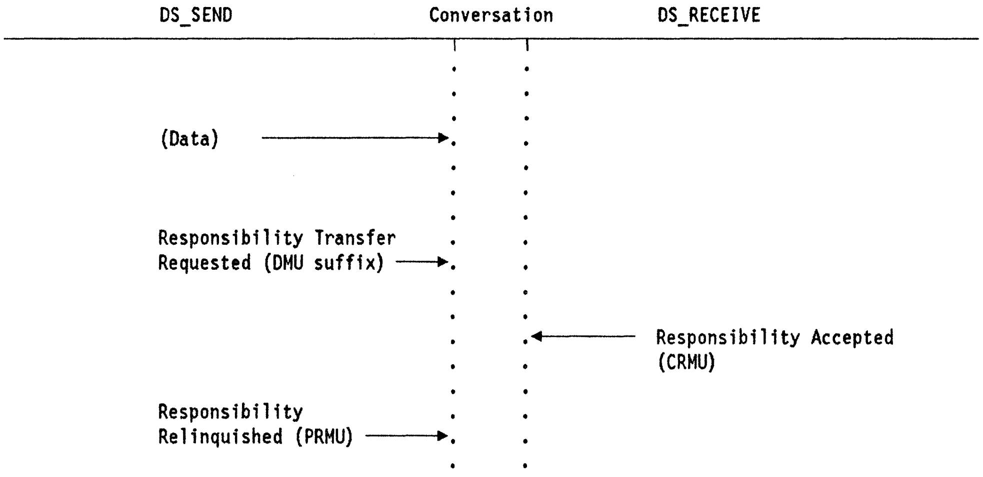


Figure 32. The Three-Step Flow for High-Integrity Distributions

Use of LU 6.2 Verbs--High Integrity

Figure 33 on page 78 shows the sequence of LU 6.2 basic conversation verbs that DS_Send and DS_Receive issue to transfer a single DMU for a high-integrity distribution. In this example, DS_Send allocates the conversation; subsequently, it retrieves a distribution from the next-DSU queue. It encodes the distribution as a DMU and issues one or more Send_Data verbs to send the DMU to DS_Receive. After sending the last portion of the DMU, DS_Send issues Receive_And_Wait to place DS_Receive in send state.

When DS_Receive is attached by DS_Send, it begins issuing Receive_And_Wait verbs to receive the DMU. It enqueues a control block representing the DMU on the mid-MU restart queue. As DS_Receive receives each buffer of data, it parses the data and updates the control block. When it receives the DMU suffix, DS_Receive enqueues the DMU control block on the router-director queue, removes it from the mid-MU restart queue, and starts DS_Router_Director. After DS_Receive successfully receives the DMU and performs the required receive-time checking (see Appendix C), it accepts responsibility for the DMU.

When DS_Receive accepts responsibility for the DMU, it builds a Completion-Report MU (CRMU) specifying that the MU_ID is COMPLETED and enqueues it on the control-MU queue. When DS_Receive is placed in send state (as a result of DS_Send's issuing Receive_And_Wait), it sends the CRMU (and all other control MUs that are waiting on the queue). The CRMU (COMPLETED) informs DS_Send that DS_Receive has accepted responsibility for the DMU. When the control-MU queue is empty, DS_Receive issues Receive_And_Wait to return to receive state.

DS_Send issues Receive_And_Wait verbs to receive the control MUs from DS_Receive. As it receives each control MU, it updates the MU_ID registry and the DMU control block to which the control MU refers. When it receives the CRMU (COMPLETED), it deletes the entry for the DMU from the next-DSU queue and deletes any server object associated with the DMU (if appropriate). After marking the MU_ID PURGED, it generates a Purge-Report MU (PRMU) and enqueues it on the control-MU queue.

When DS_Send returns to send state (as a result of DS_Receive's issuing Receive_And_Wait), it sends any queued control MUs to DS_Receive, including the PRMU. When DS_Receive receives the PRMU, it marks the MU_ID PURGED. After sending the PRMU, DS_Send may continue sending another DMU or deallocate the conversation.

Each time DS_Send enters send state (including the first time, when it begins executing) it sends all queued control MUs to DS_Receive. (This is not shown in Figure 33 on page 78. Before sending the DMU, DS_Send would send any control MUs on its queue.) When the control-MU queue is empty, DS_Send selects a distribution and sends it. After sending the distribution, DS_Send may again check the control-MU queue and send any new queued control MUs. Since control MUs, in general, allow the DSUs to resolve outstanding work items, each partner sends all its queued control MUs whenever possible. Before sending another DMU, however, DS_Send enters receive state to allow DS_Receive to send its control MUs.

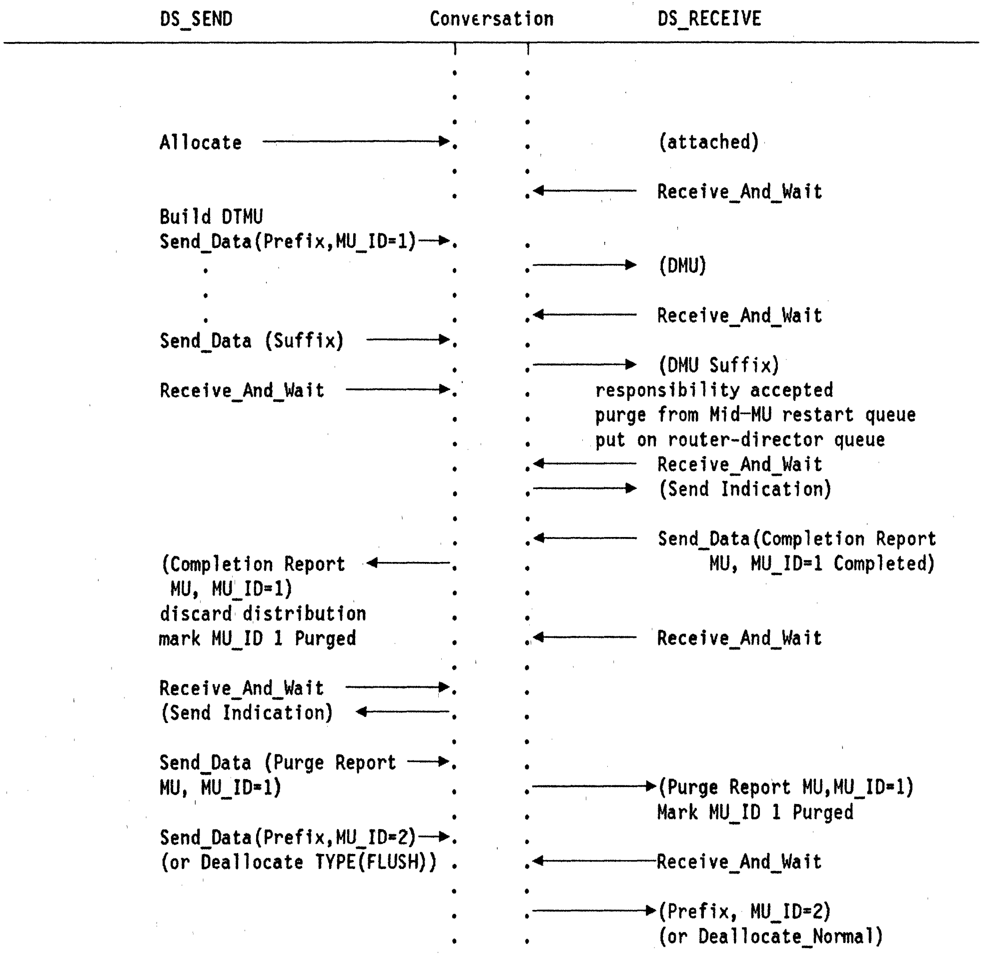


Figure 33. Protocol for Transmitting High-Integrity Distributions

Use of LU 6.2 Verbs--Basic Integrity

The verb protocol used to transfer basic-integrity DMUs is similar to that used for high-integrity DMUs. However, the DSUs do not exchange control MUs to confirm the transfer of basic-integrity traffic. After sending the suffix of a basic-integrity DMU, DS_Send discards the distribution and issues Receive_And_Wait to allow DS_Receive to send any queued control MUs. These control MUs might contain conversation control information (see "Throughput Control" on page 79) or information about other high-integrity DMUs, but DS_Receive does not generate a CRMU for the basic-integrity DMU it received from DS_Send. After sending any queued control MUs, DS_Receive issues Receive_And_Wait to return DS_Send to send state. DS_Send then continues with another DMU or deallocates the conversation. No PRMU is sent for the basic-integrity DMU.

Figure 34 on page 79 illustrates a basic-integrity transfer.

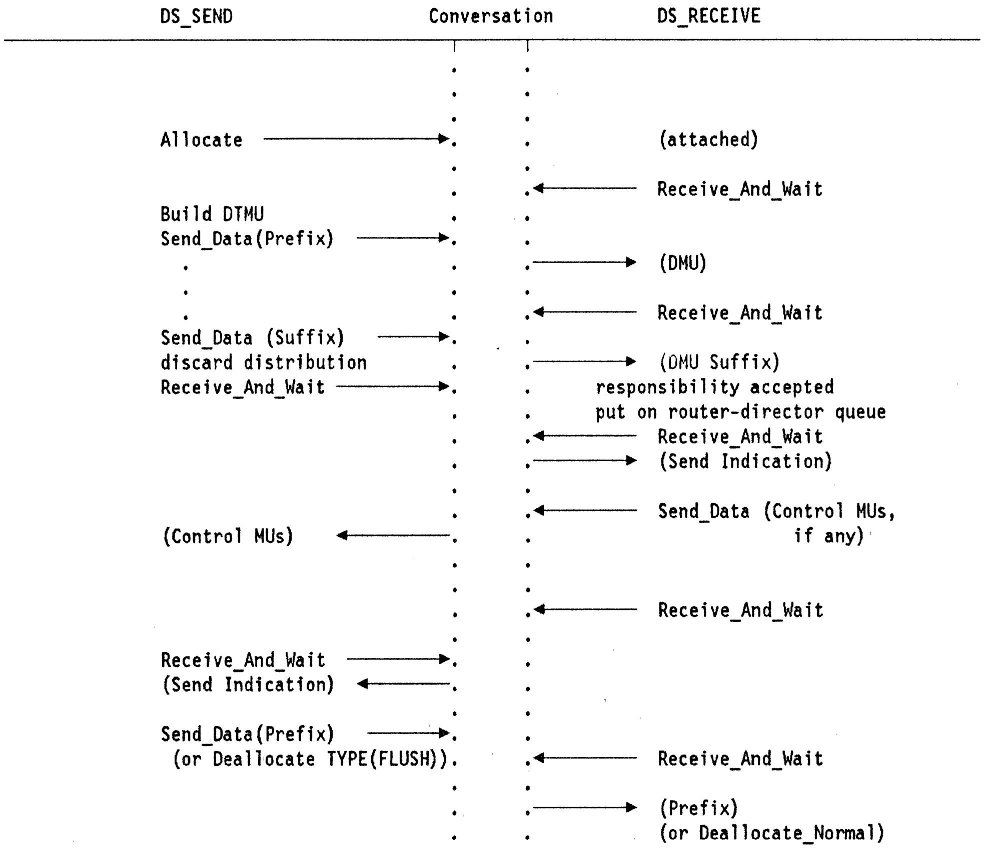


Figure 34. Protocol for Transmitting Basic-Integrity Distributions

Parallel Sessions

The examples given above show all the exchanges between DS_Send and DS_Receive occurring on a single conversation. In general, two DSUs may communicate using parallel sessions. In this environment, the control MUs pertaining to a DMU are placed on a control-MU queue and sent by the first available instance of DS_Send or DS_Receive. The CRMU and PRMU referring to a DMU may thus be sent on conversations different from the one on which the DMU was sent. In general, any control MU may flow on a conversation different from that of the DMU to which it refers.

Throughput Control

DS_Send may send an unlimited number of DMUs on a conversation before deallocating. If DS_Receive is unable to receive an unlimited number of DMUs, it may instruct DS_Send to stop sending via the *terminate_conversation* flag.

If DS_Receive returns a CRMU with the *terminate_conversation* flag set ON, DS_Send stops sending DMUs on the conversation. DS_Send may, however, continue sending control MUs until the control-MU queue is empty. When DS_Send has no more control MUs to send, it deallocates the conversation.

DS_Receive may use the *terminate_conversation* flag to achieve as low a level of throughput (DMUs received) on the conversation as it desires. For example, if DS_Receive is prepared to accept only one DMU per conversation, it sets *terminate_conversation* ON in the first CRMU it returns to DS_Send. Figure 35 on page 80 illustrates this scenario. DS_Send allocates the conversation and sends the first DMU to DS_Receive; DS_Receive returns the CRMU (COMPLETED) with *terminate_conversation* set ON. DS_Send then sends the PRMU and deallocates the conversation.

DS_Receive may return the *terminate_conversation* indication in the first CRMU it sends to DS_Send or in any subsequent CRMU.

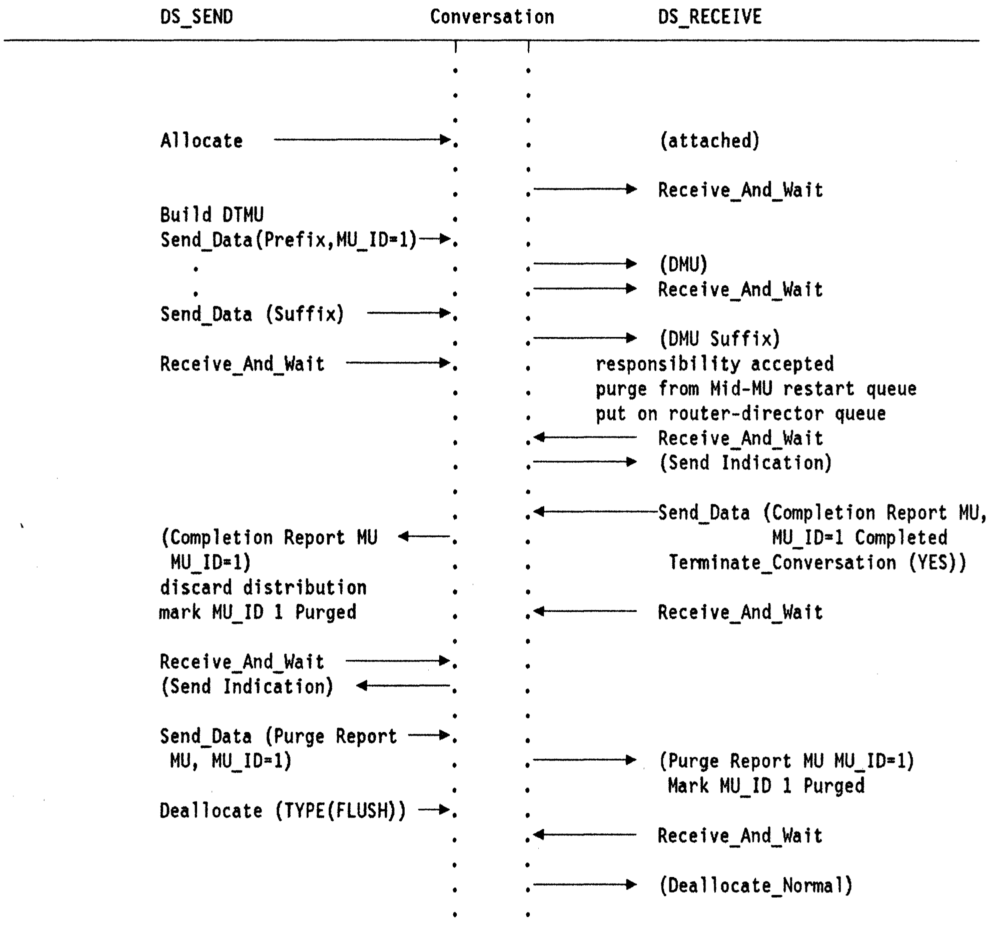


Figure 35. Use of the Terminate_Conversation Flag--One DMU Sent per Conversation

Management of Message Unit IDs

States and State Changes

A high-integrity DMU being transmitted from an instance of DS_Send to the partner instance of DS_Receive passes through several logical states. For example, while flowing over the LU 6.2 conversation, the MU is ″Being-Transmitted.″ In the period of time after it has been completely sent by DS_Send but before the transfer of responsibility messages have been exchanged, the MU is ″Pending-Transfer.″

Since the sending and receiving DSUs use MU_IDs to correlate transmission-control and exception-handling information, it is the state of an MU_ID rather than the state of an MU that is of formal interest. An MU_ID represents a particular piece of work in progress between the transport sublayers of the two DSUs. When a distribution is selected for processing by DS_Send, it is assigned a particular MU_ID; while it is being processed, the MU_ID uniquely identifies it.

The two DSUs refer to the distribution using its MU_ID until they eventually agree to terminate the ″unit of work″ represented by that MU_ID; at that time, they both mark the MU_ID PURGED. The purging of the MU_ID (i.e., the termination of the ″unit of work″), however, does not imply anything in particular about the distribution itself. The DSUs might decide to purge the MU_ID because the distribution has been successfully transferred to the receiver; because the distribution has failed and is being terminated; or because the distribution will be retried, but will be assigned a different MU_ID.

MU_IDs are used only for high-integrity distributions. Basic-integrity distributions are not assigned MU_IDs.

A DSU maintains a distinct MU_ID registry for each connection (i.e., the set of sessions it uses having the same *LU name, mode name* combination) on which it sends or receives traffic. Furthermore, the set of sequential MU_IDs represented in one MU_ID registry is used only for DMUs flowing in one direction. In other words, partner DSUs that send DMUs to one another for a particular mode name use two sets of MU_IDs to identify traffic on each mode name; one set for traffic flowing in one direction, the other set for traffic flowing in the other direction. An MU_ID, therefore, is unique for a particular direction of traffic flow on a particular connection. There is no inter-relationship between MU_ID registries for different connections, or between a DSU′s sending and receiving registries for a particular connection.

This section presents an overview of the states of an MU_ID, and the possible state transitions that may occur at both DS_Send and DS_Receive. The states used by DS_Send and DS_Receive to track MU_IDs are different, since DS_Send and DS_Receive encounter different circumstances while processing the DMU. For a formal description of the MU_ID state transitions at DS_Send and DS_Receive, refer to “Formal Description of MU_ID State Transitions” on page 102.

States of MU_IDs at DS_Send

The states of an MU_ID at DS_Send are:

- NOT_ASSIGNED

  This is the "uninitialized" or "reset" state. No association exists between the given MU_ID and any distribution.

- IN_TRANSIT

  This state is entered from NOT_ASSIGNED or SUSPENDED. When entered from NOT_ASSIGNED state, it indicates that an MU_ID has been assigned to a particular distribution. The transition from NOT_ASSIGNED to IN_TRANSIT marks the beginning of the MU transfer to the partner DSU, regardless of when the first Send_Data verb is issued. When entered from SUSPENDED state, IN_TRANSIT indicates that retransmission of the MU using the same MU_ID has begun, with a DCMU.

  Any interruption of the transmission of a DMU that is detected by the sender is accompanied by a SEMU, regardless of whether any data has actually been sent to LU 6.2. In other words, any exception condition occurring on an MU_ID in IN_TRANSIT state is reported to the partner via a SEMU, regardless of whether any portion of the DMU has actually been transmitted.

- TRANSFER_PENDING

  This state is entered from IN_TRANSIT. It indicates that the DMU suffix (which carries the "Responsibility-Transfer Requested" semantics) has been passed to LU 6.2 for transmission to the partner DSU, and that DS_Send awaits a CRMU (COMPLETED). The CRMU (COMPLETED) carries the "Responsibility Accepted" semantics.

  DS_Send may receive REMUs and unsuccessful CRMUs (e.g., CRMU (TERMINATED)) for an MU_ID that is in TRANSFER_PENDING state; DS_Send takes appropriate exception-handling actions based on the contents of the REMU or CRMU.

- CQMU_PENDING

  This is primarily a system-failure restart state. Whenever an instance of DS_Send terminates abnormally (or is aborted), some number of MU_IDs might be left in states other than NOT_ASSIGNED, SUSPENDED, or PURGED. The DSU issues CQMUs for such MU_IDs to solicit CRMUs, from which it learns the state of each MU_ID at the partner. The DSU may also issue a CQMU for an MU_ID if it has been awaiting a CRMU for an unacceptably long period of time, and assumes that the CRMU has been lost.

  While an MU_ID is in CQMU_PENDING state, DS_Send may receive CRMUs or REMUs for it. DS_Send takes appropriate recovery actions based on the contents of the CRMU or REMU.

- TERMINATION_PENDING

  This exception-processing state is entered from IN_TRANSIT, and indicates that DS_Send has detected an exception and will not retry the transmission. DS_Send informs DS_Receive of the exception, and awaits a REMU or CRMU from DS_Receive. After receiving the REMU or CRMU, DS_Send terminates the distribution, generates a distribution report (DRMU) if appropriate, marks the MU_ID PURGED, and sends a PRMU to DS_Receive.

- RETRY_PENDING

  This is also an exception recovery state entered from IN_TRANSIT. It indicates either that DS_Send detected an exception and intends to retransmit the MU, or that DS_Send received a Prog_Error indication from DS_Receive.

  Before continuing work on the DMU, DS_Send awaits a REMU or CRMU from DS_Receive. Based on the contents of the REMU or CRMU, DS_Send may decide to restart the transmission (via a DCMU), retry the MU using a different MU_ID, or terminate the distribution and generate a distribution report (if appropriate).

- SUSPENDED

  This state is entered from TRANSFER_PENDING, CQMU_PENDING or RETRY_PENDING. It indicates that a CRMU (SUSPENDED) was received, and that the transmission of the MU is to be resumed via a DCMU.

- PURGED

  This state may be entered from any state other than NOT_ASSIGNED or IN_TRANSIT. It indicates that the DSUs have finished processing the "piece of work" represented by the MU_ID, and that the association between the MU_ID and the distribution has been broken. The MU_ID will not be reused until the Next-MU_ID-Counter for the connection is reset. PURGED implies nothing about the distribution itself. The distribution may have been successfully transferred; the distribution may have been terminated and a distribution report generated (if appropriate); or the distribution may be left on the next-DSU queue to be transmitted again, using another MU_ID.

  MU_IDs in the PURGED state may be removed from the MU_ID registry.

The typical progression of MU_ID states at DS_Send in an exception-free scenario is NOT_ASSIGNED, IN_TRANSIT, TRANSFER_PENDING, PURGED.

States of MU_IDs at DS_Receive

The states of an MU_ID at DS_Receive are:

- NOT_RECEIVED

  This is the "uninitialized" state for an MU_ID at DS_Receive.

- IN_TRANSIT

  This state is entered from NOT_RECEIVED when DS_Receive receives the first part of a DTMU or DRMU, or from SUSPENDED when DS_Receive receives the first part of a DCMU. The IN_TRANSIT state indicates that the DMU is being received from the partner DSU.

- SUSPENDED

  This state is entered from IN_TRANSIT when mid-MU restart capability is available for resuming transmission of the distribution and either:

  - A retriable exception is detected by DS_Receive; or

  - A Prog_Error indication is received from DS_Send.

The partially received distribution is held in a non-volatile mid-MU restart queue, and the partially received server object is maintained in the server.

- TERMINATED

  This state may be entered from NOT_RECEIVED, IN_TRANSIT, or SUSPENDED. From NOT_RECEIVED, it is entered when DS_Receive receives a SEMU. From IN_TRANSIT, it is entered when DS_Receive detects any non-retriable exception, when DS_Receive detects a retriable exception but mid-MU restart is not appropriate, or when DS_Receive receives a Prog_Error indication from DS_Send and mid-MU restart is not appropriate. From SUSPENDED, it is entered when DS_Receive receives a SEMU indicating a non-retriable exception.

  The TERMINATED state indicates that both of the following are true:

  - DS_Receive will not accept a retransmission (DCMU) of the DMU using the same MU_ID. DS_Send will not reuse an MU_ID that DS_Receive has marked TERMINATED until the Next-MU_ID-Counter for the connection is reset (and all MU_IDs are recycled).
  - DS_Send has not yet notified DS_Receive that it has purged the MU_ID.

  DS_Receive may discard the partially received distribution and its server object, but must maintain the MU_ID and its TERMINATED status.

- COMPLETED

  This state is entered from IN_TRANSIT. It indicates that DS_Receive has accepted responsibility for the distribution and that DS_Send may discard its copy. The COMPLETED state is entered after receiving a DMU's suffix, which carries the "Responsibility-Transfer Requested" semantics.

  Before placing the MU_ID in COMPLETED state, DS_Receive must have successfully received and stored the DMU, performed at least the minimum receive-time checks, placed the distribution on the router-director queue, and scheduled DS_Router_Director. After placing the MU_ID in COMPLETED state, DS_Receive generates a CRMU.

  The receiving DSU may then forward the distribution to its next hop, deliver it locally, or generate a distribution report, as appropriate. However, the DSU must maintain the MU_ID and its COMPLETED state until the "Responsibility-Relinquished" signal is received from the partner (via a PRMU).

- PURGED

  This state is entered from SUSPENDED, TERMINATED, or COMPLETED when DS_Receive receives a PRMU. It indicates that DS_Send has placed the MU_ID in PURGED state, and thus that DS_Send will not reuse the MU_ID until the Next-MU_ID-Counter for the connection is reset. Once an MU_ID has been placed in PURGED state, it may be aged out of the MU_ID registry.

The typical progression of MU_ID states at DS_Receive in an exception-free scenario is NOT_RECEIVED, IN_TRANSIT, COMPLETED, PURGED.

MU_ID States--Active and Inactive

Each MU_ID state used by DS_Send or DS_Receive may be categorized as either active or inactive. Active states require that DS_Send or DS_Receive keep an explicit record of the MU_ID; inactive states do not require that explicit knowledge be kept. The initial and final states for both DS_Send and DS_Receive (i.e., NOT_ASSIGNED, NOT_RECEIVED and both PURGED states) are inactive; all other states are active. The MU_ID registry, therefore, contains an entry for each MU_ID in an active state. MU_IDs numbered above the current MU_ID counter are considered NOT_ASSIGNED or NOT_RECEIVED. MU_IDs numbered below the oldest MU_ID in the MU_ID registry are considered PURGED.

MU_ID States and DSU Responsibility

At any given time, exactly one DSU is responsible for a distribution. The responsible DSU must either deliver or forward the distribution, or terminate it and generate a distribution report (if appropriate). For DS_Receive, the COMPLETED state indicates that the receiving DSU is responsible for the distribution; all other active states (IN_TRANSIT, SUSPENDED, and TERMINATED) indicate that the sending DSU is responsible. DS_Receive moves from not being responsible to being responsible at the same time that the MU_ID moves from IN_TRANSIT state to COMPLETED state.

The IN_TRANSIT and SUSPENDED states clearly indicate that responsibility belongs to DS_Send. In general, however, responsibility is not as clear-cut for DS_Send as it is for DS_Receive, because DS_Send cannot determine exactly when DS_Receive accepts responsibility. For example, the MU_ID at DS_Send moves from IN_TRANSIT state to TRANSFER_PENDING state before the MU_ID at DS_Receive moves from IN_TRANSIT state to COMPLETED state, because buffering and transmission of the DMU suffix by the underlying SNA layers take a certain amount of time. While the MU_ID at DS_Receive is IN_TRANSIT, DS_Send is responsible; immediately after DS_Receive changes the MU_ID to COMPLETED, DS_Send is not responsible. In both cases, the MU_ID at DS_Send is in TRANSFER_PENDING state.

The active states at DS_Send that have a ″responsibility unclear″ connotation are TRANSFER_PENDING, TERMINATION_PENDING, RETRY_PENDING, and CQMU_PENDING. DS_Send moves to ″not responsible″ only when a ″Responsibility Accepted″ message is received from DS_Receive in the form of a CRMU (COMPLETED). Upon receipt of this message, DS_Send immediately discards the distribution and places the MU_ID in PURGED state. If DS_Receive returns any other type of REMU or CRMU, DS_Send retains responsibility and responds to the MU_ID state indicated by the REMU or CRMU.

DS_Send uses the MU_ID state to record its intentions for handling the distribution. For example, RETRY_PENDING and TERMINATION_PENDING both indicate that an exception has occurred. RETRY_PENDING indicates that DS_Send intends to retry the transmission (either by sending a DCMU or by assigning a new MU_ID and retransmitting from the beginning). TERMINATION_PENDING indicates that DS_Send will not retry the transmission, but intends to terminate it.

Before carrying out its intentions, however, DS_Send awaits a control MU (REMU or CRMU) from DS_Receive. DS_Send thus is aware of the MU_ID state at DS_Receive, and avoids inappropriate recovery actions.

MU_ID Instance Numbers

DSUs using mid-MU restart capability may transmit one or more DCMUs to complete the transmission of a DTMU. Since the DTMU and related DCMUs carry the same MU_ID, DS_Send places an instance number in each DMU. The instance number is set to 1 in the DTMU, and is incremented each time a DCMU is sent. The instance numbers allow DS_Send and DS_Receive to ignore control MUs that apply to previously sent DMUs.

DS_Send is responsible for storing the current instance number for each MU_ID and transmitting it in DMUs, SEMUs, and CQMUs. PRMUs do not carry instance numbers. If DS_Send receives a control MU containing an instance number less than the instance number currently in use for the MU_ID, it discards the control MU.

DS_Receive examines the instance number in each MU (DMU or control MU) it receives. If the instance number is greater than or equal to the current instance number in use for the MU_ID, DS_Receive processes the MU normally; if the instance number is less than the current instance number DS_Receive discards the MU. If the instance number is greater than the current instance number in use for the MU_ID, DS_Receive updates the current instance number to match the received instance number. DS_Receive returns the current instance number in each control MU (REMU or CRMU) that it sends to DS_Send.

Removing MU_IDs from the MU_ID registry

Once an MU_ID is marked PURGED, the DSUs need no longer keep an explicit entry for it in their MU_ID registries. However, an MU_ID may be removed from the registry only if all MU_IDs numbered below it have been marked PURGED (or have been previously removed from the registry). This method of removal allows the DSUs to keep track of the MU_IDs they have successfully transferred and to detect MUs that may have been lost.

For example, the receiver might find that its oldest MU_ID entry is NOT_RECEIVED, but that several higher-numbered entries have been marked PURGED. It might then assume that an MU has been lost, and issue a CRMU for the NOT_RECEIVED MU_ID to solicit its transmission. Refer to "Lost Messages" on page 97 for a further discussion of the mechanisms used to solicit MUs.

The MU_ID registry

The MU_ID registry may be viewed as a sliding window into the MU_ID space. The following example shows a registry for an *n*-session connection at DS_Receive.

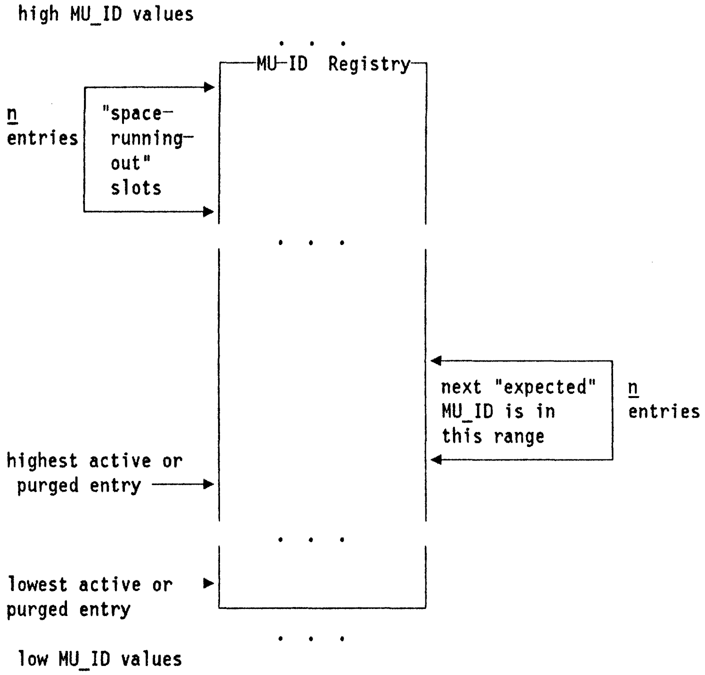

high MU_ID values
MU-ID Registry
n entries
"space-running-out" slots
next "expected" MU_ID is in this range
n entries
highest active or purged entry
lowest active or purged entry
low MU_ID values

Figure 36. The MU_ID Registry at DS_Receive for a Multiple-session Connection

The lowest numbered entries in the registry would typically be marked PURGED, representing MU_IDs on which processing has been completed but which have not yet been "aged out" of the registry. Above those entries would be entries for some number of MU_IDs that are currently being processed. These MU_IDs might be in any state, active or inactive. For example, MU_ID *x* might be COMPLETED. MU_ID *x+1*, perhaps representing a shorter DMU, might already be PURGED, and MU_ID *x+2* might be IN_TRANSIT.

Above the entries for the MU_IDs currently being processed are entries for MU_IDs that are NOT_RECEIVED. If *n* parallel sessions are active for the connection, the receiver is prepared to receive any MU_ID up to *n* higher than the highest active or purged entry. If an MU_ID higher than that is received, it indicates that an MU_ID in the "next-expected" range has been lost.

At the top of the registry are *n* entries representing a "space-running-out" area. If DS_Receive receives an MU_ID in the "space-running-out" range, it rejects the DMU and marks the MU_ID TERMINATED. The exception information returned to DS_Send indicates that the DMU was not accepted because the registry is out of space.

The most common cause of an "out-of-space" condition at DS_Receive is that a very old MU_ID remains active instead of being PURGED. Perhaps a PRMU was lost, leaving the MU_ID COMPLETED rather than PURGED, or perhaps the MU_ID is SUSPENDED and has not been resumed because of higher-priority traffic. Since the registry at DS_Receive must maintain a contiguous block of MU_ID values, and since only PURGED MU_IDs may be dropped from the bottom of the registry, an old MU_ID can prevent the registry from sliding up the MU_ID space to include higher values. The solution to this problem is for DS_Send to stop transmitting distributions and take whatever actions are necessary to move the old MU_ID into PURGED state.

Whenever DS_Receive detects the "registry-running-out-of-space" condition, it issues CRMUs for its oldest active or NOT_RECEIVED MU_IDs. If these MU_IDs are in SUSPENDED state, DS_Receive either converts them to TERMINATED (and recovers the server object and partially received distribution resources) before issuing the CRMU or notifies the operator of the condition and requests intervention.

If DS_Send receives a REMU indicating a "registry-full" condition followed by CRMU (SUSPENDED) messages, and has distributions with higher priority than those specified by the CRMUs, it purges the SUSPENDED MU_IDs, issues PRMUs for them, and retries the distributions later with new MU_IDs.

Synchronization of MU_ID Registries at Sender and Receiver

In order for a sender and a receiver to manage the traffic being transmitted between them, their MU_ID registries must be synchronized. That is, assuming there are no DMUs to transmit at a given time, the next MU_ID to be used by the sender should match the next MU_ID expected by the receiver. The sender is responsible for initiating a resynchronization protocol when the "next-MU_ID" values need to be reset. Both sender and receiver maintain a date/time stamp that indicates the "time of last reset" of their MU_ID registries.

The sender initiates an MU_ID resynchronization when a connection is first established between two DSUs. The initial conversation on the connection may be allocated by either sender or receiver, but DS_Send always initiates the resynchronization sequence. The sender also performs resynchronization when it has used all the allowed values in its MU_ID registry. In addition, the sender may initiate resynchronization in response to operator action, or if it determines that an exception has caused its registry to be out of synchronization with that of the receiver (perhaps indicated by different "time-of-last-reset" values at sender and receiver).

When the sender wishes to request re-initialization of the receiver's MU_ID registry, it stops transmitting DMUs on the connection. The sender then checks that each MU_ID in its (the sender's) MU_ID registry is in an inactive state (i.e., in either NOT_ASSIGNED or PURGED state).

In the case of a normal resynchronization (i.e., DS_Send has not detected an "out-of-synch" condition), DS_Send follows normal protocols to move active MU_IDs to PURGED state. If DS_Send has detected an "out-of-synch" condition, it notifies operations. Operations then decides on the appropriate action to take for each active MU_ID. For MU_IDs in a "responsibility-unclear" state. operations would typically change the MU_ID to PURGED state, but retain the distribution on the next-DSU queue for retransmission later.

After checking that all MU_IDs in its registry are in NOT_ASSIGNED or PURGED state, the sender transmits a Reset-Request MU. The Reset-Request MU contains the MU_ID that the sender will transmit next--i.e., the value that the receiver should expect next--and a time stamp. The sender stores the time stamp as the "time of last reset" of its MU_ID registry.

Upon receiving the Reset-Request MU, the receiver checks to see whether any MU_IDs in its registry are in an active state (any state other than NOT_RECEIVED or PURGED). If it finds any such MU_IDs, the receiver refuses the Reset Request MU by returning CRMUs indicating the state of each active MU_ID. The sender responds to each CRMU with a PRMU, to allow the receiver to mark all MU_IDs PURGED. After responding to all the CRMUs, the sender sends another Reset-Request MU with a new time stamp.

If, upon receipt of the Reset-Request MU, the receiver finds that all MU_IDs in its registry are in inactive states, it re-initializes its MU_ID registry such that its "next-expected" MU_ID is the MU_ID value sent in the Reset-Request MU. It stores the time stamp from the Reset-Request MU as the "time of last reset" of its MU_ID registry. The receiver then transmits a Reset-Accepted MU to the sender. In the Reset-Accepted MU, it echoes the MU_ID and time stamp from the Reset-Request MU.

When the sender receives a Reset-Accepted MU in response to its last Reset-Request MU (indicated by the time stamp in the Reset-Accepted MU matching the time stamp stored by the sender) it re-initializes its MU_ID registry such that its "next-to-be-sent" MU_ID matches the value in the Reset-Accepted MU. If the sender receives a Reset-Accepted MU whose time stamp does not match the sender's time stamp, it discards the MU.

The "time-of-last-reset" time stamps may be used to detect out-of-synchronization conditions between the two MU_ID registries. For example, when DS_Receive receives an MU_ID outside the range of MU_IDs that it is prepared to accept (i.e., the MU_ID is either too high or too low), it returns a REMU containing the MU_ID it received, the MU_ID it expected, and the time stamp of its MU_ID registry. DS_Send can then compare the time stamp with its own "time of last reset." If the time stamps do not match, DS_Send assumes that the registries are out of synchronization and initiates resynchronization.

Exceptions Detected by the Distribution Transport Sublayer

Exceptions Detected by DS_Send

DS_Send informs DS_Receive that it has detected an exception by issuing a Send_Error verb to LU 6.2 and sending a Sender-Exception Message Unit (SEMU). The processing performed by DS_Send and DS_Receive in response to a sender-detected exception is described below.

The Sender-Exception Message Unit (SEMU)

The SEMU is sent from DS_Send to DS_Receive. It is used to inform DS_Receive that DS_Send has detected an exception while processing a particular DMU. SEMUs are always generated for sender-detected exceptions that involve high-integrity DMUs. A SEMU may be generated for a basic-integrity DMU, but is not required. A SEMU contains the MU_ID of the (high-integrity) DMU to which it pertains, along with a report code describing the exception condition. Appendix E lists the report codes used by DS.

Any exception that occurs after an MU_ID has been assigned to a particular distribution is reported to DS_Receive via a SEMU, whether DS_Send has begun transmitting the DMU or not.

DS_Send builds a SEMU at the time it detects an exception. DS_Send may also generate a SEMU if it detects that a previously issued SEMU has been lost. For example, suppose that DS_Receive inquires, via a CRMU, about an MU_ID that it has not received. If DS_Send has placed that MU_ID in TERMINATION_PENDING state, it assumes that the SEMU it previously sent was lost; DS_Send then rebuilds an identical SEMU and sends it to DS_Receive. See "Actions of DS_Send in Response to a CRMU" on page 95 for a description of DS_Send's actions on receiving a CRMU.

Effects of a Sender-Detected Exception on DS_Send

When DS_Send detects an exception, it first determines whether the exception is recoverable or nonrecoverable. Exceptions that are inherently recoverable may be considered nonrecoverable after an implementation-defined number of transmission retries have been attempted.

If the exception is inherently recoverable, DS_Send places an exception-hold on the next-DSU queue for the connection. This hold condition precludes all currently active instances of DS_Send from retrieving distributions for transmission. The exception-hold condition may be lifted by operator intervention or by the attaching of a new instance of DS_Send by DS_Receive at the partner DSU. If the exception is inherently nonrecoverable, DS_Send does not place a hold on the next-DSU queue.

If DS_Send is not currently processing a distribution (or if it has finished sending an entire distribution), it takes no further action. If DS_Send is processing a high-integrity distribution, it places the MU_ID in either TERMINATION_PENDING or RETRY_PENDING state depending on whether the exception is nonrecoverable or recoverable, respectively. The state of the MU_ID indicates the intentions of DS_Send (i.e., either termination of the distribution or retransmission) pending knowledge of the state of the MU_ID at DS_Receive.

If DS_Send is processing a basic-integrity distribution, it discards the distribution if the exception is unrecoverable. If the exception is recoverable, DS_Send leaves the distribution on the queue to be retried later.

If DS_Send has been reading an object from the server during transmission, it issues a verb to the server to terminate its access to the object.

DS_Send then informs DS_Receive of the exception. If it has begun transmitting the DMU to DS_Receive, it informs DS_Receive by issuing Send_Error. If DS_Send has not begun transmitting the DMU, it does not issue Send_Error. For a high-integrity DMU, DS_Send builds a SEMU (whether it has begun transmission of the DMU or not) describing the exception condition and enqueues it on the Control MU queue. For a basic-integrity DMU, DS_Send may generate a SEMU but is not required to.

Finally, DS_Send logs the exception condition. It leaves the distribution on the next-DSU queue for further processing.

Actions of DS_Receive in Response to Send_Error

The Send_Error verb issued by DS_Send results in the receipt of either a Prog_Error_Trunc or a Prog_Error_No_Trunc indication by DS_Receive. When DS_Receive receives the Prog_Error indication on a high-integrity DMU, its actions depend on whether or not mid-MU restart is appropriate.

If DS_Receive decides that mid-MU restart is appropriate, it saves the partially received DMU on its mid-MU restart queue. It saves the partially received server object in the server space. It places the MU_ID in SUSPENDED state, pending receipt of the continuation of the DMU (i.e., pending receipt of a DCMU).

If mid-MU restart is not appropriate, DS_Receive discards the partially received DMU and its server object. It places the MU_ID in TERMINATED state. If DS_Send attempts retransmission, it will do so using a new MU_ID.

When DS_Receive receives a Prog_Error on a basic-integrity DMU, it discards the partially received DMU.

Actions of DS_Receive in Response to a SEMU

When DS_Receive receives a SEMU, the actions it takes depend on the state of the MU_ID specified in the SEMU. If the MU_ID is NOT_RECEIVED, DS_Receive changes it to TERMINATED. If the MU_ID is SUSPENDED, DS_Receive examines the exception code in the SEMU. If the exception is inherently recoverable, the MU_ID is left in SUSPENDED state; if not, DS_Receive discards the partially received DMU (and any object associated with it) and changes the state of the MU_ID to TERMINATED. If the MU_ID is in any other state, the state is left unchanged.

After performing the above actions, as appropriate, DS_Receive generates a CRMU to inform DS_Send of the state of the MU_ID (and the restart position, if the MU_ID is SUSPENDED). Finally, DS_Receive logs the information received in the SEMU.

If DS_Receive receives a SEMU that does not contain an MU_ID (probably generated while processing a basic-integrity DMU) it logs the receipt of the SEMU.

Actions of DS_Send in Response to a Conversation Failure

If DS_Send detects a conversation failure while transmitting a control MU, it leaves the control MU on the queue to be sent later. If the conversation failure occurs while DS_Send is transmitting a high-integrity DMU, DS_Send places the MU_ID in RETRY_PENDING state, terminates its access to the distribution and the server object, and generates a SEMU. The SEMU initiates recovery protocols with DS_Receive. If DS_Send is transmitting a basic-integrity DMU, it simply leaves the DMU on the queue to be retried later.

Actions of DS_Send in Response to an Operator Purge

An operator-initiated purge of an already-started distribution (i.e., a distribution that has already been assigned an MU_ID) is handled like a nonrecoverable exception. DS_Send issues Send_Error followed by a SEMU, and places the MU_ID in TERMINATION_PENDING state. Following the receipt of a CRMU (TERMINATED), DS_Send will purge the distribution and issue a PRMU.

Exceptions Detected by DS_Receive

DS_Receive informs DS_Send that it has detected an exception by issuing a Send_Error verb to LU 6.2 PS and sending a Receiver-Exception Message Unit (REMU). The processing performed by DS_Send and DS_Receive in response to a receiver-detected exception is described below.

The Receiver-Exception Message Unit (REMU)

The REMU is sent from DS_Receive to DS_Send. It is used to inform DS_Send that DS_Receive has detected an exception condition while processing a particular DMU. A REMU contains the MU_ID (for high-integrity traffic) of the DMU to which it pertains, an *SNA_Condition_Report* describing the exception condition, and certain other fields used by DS_Send to respond to the exception.

The REMU is used to report both recoverable and nonrecoverable exceptions. DS_Receive always generates REMUs for receiver-detected exceptions that involve high-integrity DMUs. DS_Receive may generate REMUs for exceptions involving basic-integrity traffic but is not required to do so.

Effects of a Receiver-Detected Exception on DS_Receive

When DS_Receive detects an exception, it first informs DS_Send by issuing Send_Error (assuming the conversation is still available-- if the conversation has failed, it cannot issue Send_Error). For high-integrity DMUs, the actions of DS_Receive following the Send_Error depend on whether the exception condition is recoverable, and whether or not mid-MU restart is appropriate.

If the exception is recoverable and DS_Receive decides that mid-MU restart is appropriate, DS_Receive issues a Terminate_Write verb to terminate its access to the partially received server object. It leaves the partially received DMU on the mid-MU restart queue. It places the MU_ID in SUSPENDED state, pending continuation of the transmission.

If the exception is not recoverable or if DS_Receive decides not to attempt mid-MU restart, DS_Receive discards the partially received DMU and its server object. It places the MU_ID in TERMINATED state; if DS_Send attempts retransmission, it will do so using a different MU_ID.

Finally, DS_Receive enqueues a REMU describing the exception condition on its control-MU queue, and logs the exception.

For exceptions involving basic-integrity DMUs, DS_Receive simply discards the partially received DMU after issuing Send_Error.

Actions of DS_Send in Response to Send_Error

The Send_Error verb issued by DS_Receive results in the receipt of a Prog_Error_Purging indication by DS_Send. When DS_Send receives Prog_Error_Purging, it places the MU_ID in RETRY_PENDING state, pending the receipt of the REMU from DS_Receive. If DS_Send has been reading a server object during transmission, it issues a server verb to terminate its access to the object; the server verb instructs the server to retain its mid-MU restart checkpoint information (if the server provides mid-MU restart capability). DS_Send leaves the distribution on the next-DSU queue for further processing. If DS_Send receives a Prog_Error_Purging indication while sending a basic-integrity DMU, it discards the DMU.

Actions of DS_Send in Response to a REMU

The actions taken by DS_Send in response to a REMU depend on the type of exception reported in the REMU. If the exception is not recoverable, DS_Send terminates the distribution (and any associated object), marks the MU_ID PURGED, sends a PRMU to DS_Receive, and generates a distribution report, if appropriate, to inform the report-to DSU or user that the distribution was not delivered.

If the exception is inherently recoverable, DS_Send places an exception-hold on all next-DSU queues for the connection. The hold will be removed by operator action or by DS_Receive's attaching another instance of DS_Send.

If the exception is recoverable but the retry count has been exhausted for the DMU, DS_Send then treats the exception just as it would a non-recoverable exception.

If the exception is recoverable and DS_Send decides to attempt mid-MU restart, DS_Send issues a CQMU to determine the location in the DMU at which retransmission should begin. If the exception is recoverable but DS_Send decides not to attempt mid-MU restart, DS_Send marks the MU_ID PURGED, sends a PRMU so that DS_Receive can mark the MU_ID PURGED, and leaves the distribution on the next-DSU queue so that it can be retried later, with a different MU_ID.

However, if DS_Receive encounters a recoverable exception and DS_Send encounters a non-recoverable exception, DS_Send proceeds as it would for the non-recoverable exception. For example, suppose that DS_Send determines that the exception reported in the REMU is recoverable, but finds that the MU_ID has previously been placed in TERMINATION_PENDING state. After receiving the REMU, DS_Send would terminate the distribution.

If the REMU indicates that the received MU_ID is a duplicate, DS_Send checks the MU_ID registry information returned in the REMU. If the MU_ID registries are synchronized, DS_Send discards the duplicate DMU. If the MU_ID registries are out of synchronization, DS_Send initiates a resynchronization sequence (see "Synchronization of MU_ID Registries at Sender and Receiver" on page 88).

After completing the processing of the REMU as appropriate, DS_Send logs the receipt of the REMU.

If both DS_Send and DS_Receive detect a conversation failure while processing a particular MU_ID, DS_Send may receive a REMU informing it of a conversation failure of which it is already aware. In such a case, DS_Send discards the redundant REMU.

If DS_Send receives a REMU that does not contain an MU_ID (probably generated while processing a basic-integrity DMU) it logs the receipt of the REMU.

Actions of DS_Receive in Response to a Conversation Failure

If DS_Receive detects a conversation failure while receiving a high-integrity DMU, its actions depend on whether mid-MU restart is appropriate. If it decides that mid-MU restart is appropriate, DS_Receive saves the partially received distribution and server object and places the MU_ID in SUSPENDED state. Otherwise, it discards the distribution and server object and places the MU_ID in TERMINATED state. In either case, it generates a REMU and places it on the control-MU queue. The REMU notifies DS_Send of the particular MU_ID that was interrupted by the conversation failure.

If DS_Receive detects a conversation failure while receiving a basic-integrity DMU, it discards the partially received DMU.

Other Control MUs (CQMU, CRMU, PRMU)

DS_Send and DS_Receive use three other types of control MUs to exchange information about high-integrity DMUs. These MUs are not used to exchange information about basic-integrity MU_IDs (although DS_Receive may use the CRMU to send conversation control information (i.e., the *terminate_conversation* flag) while receiving basic-integrity traffic).

Completion Query Message Unit (CQMU)

The CQMU is sent from DS_Send to DS_Receive. DS_Send uses the CQMU to inquire about the state of a particular MU_ID at the adjacent DSU. A CQMU contains the MU_ID about which DS_Send is inquiring. A CQMU requests DS_Receive to return a CRMU describing the state of the MU_ID.

CQMUs are not sent in normal exception-free situations. DS_Send issues an implicit request for a CRMU by sending a (high-integrity) DMU suffix or a SEMU; if the CRMU arrives in a timely manner, DS_Send has no need to send a CQMU. If the CRMU does not arrive in a timely manner, DS_Send issues a CQMU to solicit another CRMU.

DS_Send may also generate a CQMU in response to a REMU. If the exception reported in the REMU is recoverable, and if DS_Send wishes to attempt mid-MU restart, it issues a CQMU to determine the location at which it should begin its retransmission.

A CQMU may also be generated if the sending DSU is restarted after a failure; the DSU may send a CQMU for any MU_ID if it is unsure of the state of that MU_ID at DS_Receive.

Actions of DS_Receive in Response to a CQMU

If DS_Receive is able to find the entry in its MU_ID registry for the MU_ID specified in the CQMU, it simply generates a CRMU to inform DS_Send of the state of that MU_ID. If the state of the MU_ID is SUSPENDED, the CRMU also contains an indication of the point at which retransmission should begin.

If DS_Receive is not able to find the entry in its MU_ID registry because the entry has been aged out, it returns a CRMU indicating that the MU_ID has been PURGED. If DS_Receive is not able to find the entry in its MU_ID registry because the entry has not yet been used, it returns a CRMU indicating that the MU_ID is NOT_RECEIVED.

Completion Report Message Unit (CRMU)

The Completion Report Message Unit (CRMU) is sent from DS_Receive to DS_Send. It is used by DS_Receive to inform DS_Send of the state of a particular MU_ID, and to control the traffic flow on the conversation. A CRMU may contain an MU_ID and flags indicating the state of that MU_ID at DS_Receive. For an MU_ID in SUSPENDED state, the CRMU also contains the ID of the last high-level LLID structure received and the byte count of the last byte of that structure that was received.

DS_Receive also uses the CRMU to instruct DS_Send to stop sending traffic on a conversation. See "Throughput Control" on page 79 for further details.

DS_Receive generates a CRMU whenever it receives a request for one from DS_Send. A request for a CRMU may be either explicit or implicit. DS_Send explicitly requests a CRMU by sending a CQMU; DS_Receive responds to the CQMU by generating a CRMU. DS_Send implicitly requests a CRMU by finishing a particular transmission of a (high-integrity) DMU, that is, by sending either the suffix of a DMU or a SEMU. DS_Receive responds to a DMU suffix or a SEMU by generating a CRMU.

DS_Receive may also generate a CRMU if it detects that a transmission from DS_Send has been lost. See "Lost Messages" on page 97 for further information on lost messages.

Actions of DS_Send in Response to a CRMU

A CRMU with MU_ID state COMPLETED acts as a confirmation to DS_Send that DS_Receive has accepted responsibility for the DMU. When DS_Send receives the CRMU (COMPLETED), it discards its copy of the distribution and marks the MU_ID PURGED. After doing so, it informs DS_Receive by sending a Purge-Report MU (PRMU). The PRMU informs DS_Receive that it may mark the MU_ID PURGED, since DS_Send will not use that MU_ID in subsequent transmissions.

A CRMU with MU_ID state TERMINATED informs DS_Send that an exception was encountered during the processing of the DMU, and that mid-MU restart is not possible. If DS_Send has previously placed the MU_ID in TERMINATION_PENDING state, it terminates the distribution, generates a distribution report if appropriate, marks the MU_ID PURGED, and sends a PRMU to DS_Receive. If the MU_ID is in any other state, DS_Send retries the transmission with a new

MU_ID. It marks the current MU_ID PURGED, sends a PRMU, and leaves the distribution on the next-DSU queue to be retried later, using a different MU_ID.

A CRMU with MU_ID state SUSPENDED informs DS_Send that an exception was encountered during the processing of the DMU, and that from DS_Receive's point of view, mid-MU restart is possible. If DS_Send has placed the MU_ID in TERMINATION_PENDING state, it terminates the distribution, generates a distribution report if appropriate, marks the MU_ID PURGED, and sends a PRMU to DS_Receive. If the MU_ID is in any other state and DS_Send wishes to attempt recovery using mid-MU restart protocols, it saves the restart position indicated in the CRMU and places the MU_ID in SUSPENDED state. DS_Send may then begin retransmission at a later time. If DS_Send chooses to re-send the entire DMU rather than to attempt recovery using mid-MU restart protocols, it marks the MU_ID PURGED, sends a PRMU, and leaves the distribution on the next-DSU queue; it will be retried later using another MU_ID.

A CRMU with MU_ID state IN_TRANSIT informs DS_Send that the indicated DMU is still being received. DS_Send ignores the CRMU unless it has placed the MU_ID in TERMINATION_PENDING or RETRY_PENDING states; for an MU_ID in either of those states, DS_Send issues a CQMU to solicit another CRMU.

A CRMU with MU_ID state NOT_RECEIVED informs DS_Send that DS_Receive has not received any part of the indicated DMU, nor has it received a SEMU indicating an error. DS_Send ignores the CRMU if the MU_ID is in TRANSFER_PENDING state; it issues a SEMU if the MU_ID is in CQMU_PENDING or PURGED state. If the MU_ID is in TERMINATION_PENDING or RETRY_PENDING state, DS_Send assumes the SEMU it sent earlier has been lost, and re-sends an identical SEMU.

Purge-Report Message Unit (PRMU)

The Purge-Report Message Unit (PRMU) is sent from DS_Send to DS_Receive. It informs DS_Receive that DS_Send has marked a particular MU_ID PURGED. The PRMU contains the MU_ID that has been purged.

DS_Send never sends a PRMU unless DS_Receive has solicited it via a REMU or CRMU. DS_Send may mark an MU_ID PURGED either because DS_Receive has accepted responsibility for the DMU or because an exception has occurred and DS_Send has decided not to attempt recovery using the same MU_ID.

Actions of DS_Receive in Response to a PRMU

The PRMU informs DS_Receive that DS_Send has marked the specified MU_ID PURGED, and therefore that DS_Send has terminated the piece of work represented by that MU_ID. Consequently, DS_Receive need no longer retain an explicit memory of the state of that piece of work. If the state of that MU_ID at DS_Receive is anything other than SUSPENDED, DS_Receive simply marks the MU_ID PURGED. DS_Receive may then "age out" the entry for the MU_ID from its MU_ID registry. If the state of the MU_ID is SUSPENDED, DS_Receive discards its partial copy of the DMU as well as marking the MU_ID PURGED.

The PRMU does not imply anything in particular about the disposition of the distribution itself, merely that the MU_ID has been marked PURGED. The distribution may have been successfully transferred, it may have been terminated and a distribution report generated, or DS_Send may have decided to retry it with a different MU_ID.

Lost Messages

The transfer of a high-integrity distribution from one DSU to another normally involves three messages. The sender sends the DMU itself; the receiver answers by sending a CRMU; the sender sends the PRMU. So long as none of these MUs is lost, their exchange keeps MU_IDs staging through each DSU's MU_ID registry. Each MU_ID moves from NOT_ASSIGNED (or NOT_RECEIVED) state to (eventually) PURGED state, and the oldest MU_IDs are "aged out" of the registry.

If one of the three MUs (DMU, CRMU, or PRMU) is lost, the result is an MU_ID that does not move to PURGED state in a timely manner. In particular, a DSU assumes that an MU has been lost if it finds an inappropriately "old" entry in its MU_ID registry for an MU_ID in an active state. If the state of the MU_ID indicates that a response regarding the MU_ID is expected from the partner, and if that response is not forthcoming, the DSU attempts to solicit the response.

At DS_Send, the MU_ID states that indicate that a response is expected from the partner are TRANSFER_PENDING, CQMU_PENDING, TERMINATION_PENDING, and RETRY_PENDING. If the DSU finds that an MU_ID remains in one of these states for an inappropriately long period of time, it issues a CQMU for the MU_ID. The CQMU requests DS_Receive to return a CRMU indicating the state of the MU_ID at DS_Receive. Upon receipt of the CRMU, DS_Send is able to continue work on the MU_ID as appropriate.

At DS_Receive, the MU_ID states that indicate that a response is expected from DS_Send are SUSPENDED, TERMINATED, and COMPLETED. In addition, the NOT_RECEIVED state may indicate that a DMU is expected from the partner, if DMUs with MU_IDs numbered above the NOT_RECEIVED MU_ID have already been received and marked PURGED. If an MU_ID remains in a "response-pending" state for too long, DS_Receive issues a CRMU indicating the state of the MU_ID. The CRMU requests DS_Send to send a DMU, SEMU, or PRMU, as appropriate.

In general, the objective of each DSU is to reduce its number of "to-be-purged" MU_IDs, while at the same time exercising restraint so as not to flood the connection with CQMUs or CRMUs. The period of time that an MU_ID is allowed to remain in a "response-pending" state varies with the properties of the connection.

On a single-session connection, for example, only two MU_ID entries might be needed in the receiver's registry: one marked NOT_RECEIVED, the other marked PURGED. If an MU_ID greater than the current NOT_RECEIVED MU_ID were sent by DS_Send, DS_Receive could reject it and issue a CRMU (NOT_RECEIVED) to solicit the MU_ID it expected.

On a busy parallel-session connection, on the other hand, work might proceed on many MU_IDs concurrently. DS_Receive could find an entry in its MU_ID registry in NOT_RECEIVED state, even if several entries numbered above that entry had already been received and marked PURGED. The NOT_RECEIVED MU_ID might simply have been delayed while making its way across one of the parallel sessions. DS_Receive would therefore wait longer before issuing the CRMU (NOT_RECEIVED) on the parallel-session connection than on the single-session connection.

DS_Send solicits lost messages with CQMUs and DS_Receive solicits lost messages with CRMUs. DS_Send and DS_Receive may issue as many CQMUs and CRMUs as necessary, until a response is received from the partner. Since DS_Receive replies to a CQMU by sending a CRMU, and since DS_Send typically replies to a CRMU by sending a PRMU (even if the MU_ID has previously been purged, or even aged out of the registry), duplicate CQMUs or CRMUs cause no harm. Each DSU exercises appropriate restraint in sending CQMUs and CRMUs, however, to avoid flooding the connection with control messages.

Mid-MU Restart

If an exception occurs during the transmission of a DTMU or DCMU, and if DS_Receive has successfully received the DMU at least through the *destination_list*, DS_Receive may inform DS_Send that mid-MU restart is possible. The use of mid-MU restart allows the two DSUs to resume the failed transmission at or near the point of failure.

The receiver uses the *last_structure_received* and the *last_byte_received* fields in the CRMU to inform the sender of the portion of the DMU that was successfully received. If the receiver supports only LLID restart (i.e., it supports restart only at LLID boundaries, not within LLIDs), it informs the sender of the last high-level LLID structure that was completely received by specifying the ID of that structure in *last_structure_received.* It indicates that the structure was received in its entirety in one of two ways: either by specifying a value of X'FFFFFFFFFFFFFFFF' for *last_byte_received* or by omitting the *last_byte_received* field entirely (the two methods are equivalent).

If the receiver supports the byte-count restart elective, it may choose to inform the sender that restart within a structure is possible. It does so by specifying the ID of the last high-level LLID structure that was partially received in *last_structure_received.* It indicates the number of bytes of that structure that were successfully received (not including the LLID header bytes) in the *last_byte_received* field.

Upon receipt of the CRMU containing mid-MU restart information, DS_Send builds a DCMU to continue the transmission of the distribution. If the CRMU indicates that the *last_structure_received* was received in its entirety (i.e., if *last_byte_received* is omitted or has a value of X'FFFFFFFFFFFFFFFF'), DS_Send is expected to build a DCMU that begins with the structure immediately following the *last_structure_received.* However, DS_Send may choose to restart with an earlier LLID structure.

If the CRMU indicates a restart position within the *last_structure_received* (i.e., if *last_byte_received* is present and contains a value other than X'FFFFFFFFFFFFFFFF'), and if DS_Send is capable of byte-count restart, DS_Send builds a DCMU that begins with the portion of the *last_structure_received* immediately following the *last_byte_received*. If DS_Send does not support the byte-count restart elective, it ignores the value in *last_byte_received* and restarts at the beginning of the *last_structure_received*.

DS_Receive accepts a DCMU restarting the distribution at the position that was indicated in the CRMU. If the CRMU indicated that restart should begin at a particular LLID boundary, DS_Receive also accepts a DCMU restarting at a different (earlier) LLID boundary. If the DCMU begins with an earlier LLID structure, DS_Receive receives and discards the redundant transmission of the earlier structure(s).

If the CRMU indicated that restart should begin at a particular byte location within the *last_structure_received*, DS_Receive accepts either a DCMU restarting at that location or a DCMU restarting at the beginning of the *last_structure_received*. If the DCMU indicates a restart position within the *last_structure_received* different from that specified in the CRMU, DS_Receive may reject the DCMU.

Example

Figure 37 on page 100 illustrates the use of control MUs to accomplish mid-MU restart following a conversation failure. DS_Send allocates the conversation and begins sending, issuing verbs to the server to read the server object. Before DS_Receive receives the DTMU suffix, a conversation failure occurs.

After allocating a new conversation (or on another already-active conversation), DS_Send issues a CQMU to query the state of the MU_ID at DS_Receive. DS_Receive issues Query_Last_Byte_Received to its server to determine the last byte of the server object that was received. DS_Receive then transmits a CRMU specifying *last_structure_received* as SERVER_OBJECT and *last_byte_received* as the byte number supplied by the server. If DS_Receive has not received any part of the server object or if the receiving DSU does not support the byte-count restart elective, DS_Receive returns the ID of the last high-level LLID structure it has fully received (either the agent object or an unrecognized field) in *last_structure_received*, and indicates that the structure was fully received either by specifying a value of X'FFFFFFFFFFFFFFFF' for *last_byte_received* or by omitting the *last_byte_received* field entirely.

Upon receipt of the CRMU, DS_Send issues server verbs specifying restart information to begin reading the server object at the byte following the byte specified by *last_byte_received*. DS_Send encodes a DCMU containing the remainder of the server object and sends it to DS_Receive. DS_Receive issues server verbs specifying restart information to store the remainder of the server object, and, upon receipt of the DTMU suffix, accepts responsibility for the distribution. DS_Receive then transmits a CRMU (COMPLETED) and receives a PRMU.

If the CRMU (SUSPENDED) specifies a *last_byte_received* value other than X'FFFFFFFFFFFFFFFF' but DS_Send does not support the byte-count restart elective, DS_Send ignores the *last_byte_received* value and restarts at the beginning of the server object.

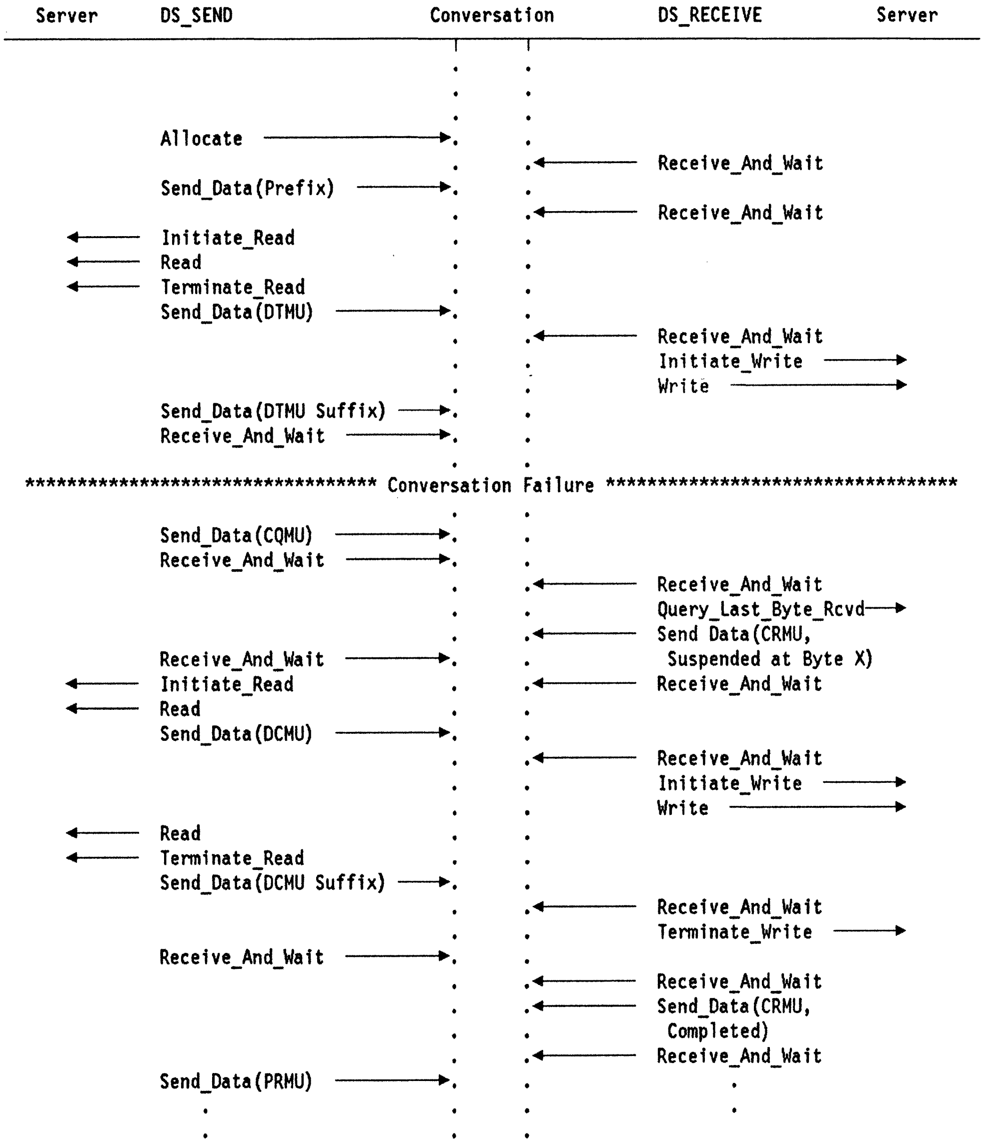


Figure 37. Example--Mid-MU Restart

Formal Description of MU_ID State Transitions

DS_SEND_MU_STATE_DESCRIPTION

Function: This finite-state machine describes the MU_ID state transitions that occur at DS_Send. Actions accompanying the state changes (such as generating a Distribution Report for a distribution entering Purged state) are not given here. See the FSMs describing DS_Send for further detail.

Notes: The CRMU input signals indicate that a CRMU was received with the indicated MU_ID state. If the MU_ID state in the CRMU is SUSPENDED, processing differs depending on whether or not Mid-MU Restart is appropriate. The SEMU input signals indicate that DS_Send encountered an exception (retriable or not, as indicated) and generated a SEMU. The REMU input signals indicate that a REMU was received indicating an exception (retriable or not, as indicated).

	States							
	NOT ASS'N'D	IN TRAN	TRANSF PEND	CQMU PEND	TERM PEND	RETRY PEND	SUS-PENDED	PURGED
Inputs	01	02	03	04	05	06	07	08
MU ID ASSIGNED	2	/	/	/	/	/	/	/
SUFFIX SENT	/	3	/	/	/	/	/	/
RESENDING WITH DCMU	/	/	/	/	/	/	2	/
FAILURE RECOVERY STARTUP	-	4	4	-	-	4	-	-
CRMU-NOT RECEIVED	-	-	-	-	-	-	/	/
CRMU-IN TRANSIT	/	/	-	-	-	-	/	-
CRMU-SUSPENDED MID MU RESTART	/	/	7	7	8	7	-	-
CRMU-SUSPENDED NO MID MU RESTART	/	/	8	8	8	8	-	-
CRMU-TERMINATED	/	/	8	8	8	8	8	-
CRMU-COMPLETED	/	/	8	8	8	8	/	-
CRMU-PURGED	/	/	/	/	/	/	/	-
CONVERSATION FAILURE RETRY	/	6	/	/	/	/	/	/
CONVERSATION FAILURE NO RETRY	/	5	/	/	/	/	/	/
SEMU-NO RETRY	/	5	/	/	/	/	/	/
SEMU-RETRY MID MU RESTART	/	6	/	/	/	/	/	/
SEMU-RETRY NO MID MU RESTART	/	6	/	/	/	/	/	/
SEMU-RETRY EXHAUSTED	/	5	/	/	/	/	/	/
PROG ERROR RECEIVED	/	6	/	/	/	/	/	/
REMU-DUPLICATE CONV FAIL REPORT	/	/	/	/	-	-	/	-
REMU-NO RETRY	/	/	8	8	8	8	/	-
REMU-RETRY MID MU RESTART	/	/	4	-	8	4	/	-
REMU-RETRY NO MID MU RESTART	/	/	8	8	8	8	/	-
REMU-RETRY EXHAUSTED	/	/	8	8	8	8	/	-

DS_RCV_MU_STATE_DESCRIPTION

Function:	This finite-state machine describes the MU_ID state transitions that occur at DS_Receive. Actions accompanying the state changes are not given here. See the FSMs describing DS_Receive for further detail.
Notes:	The After Failure Restart, Conv Fail, and Prog Err Recvd input signals indicate the exception condition that was detected and whether or not Mid-MU Restart was appropriate. The SEMU input signals indicate that a SEMU was received indicating an exception (retriable or not, as indicated). The REMU input signals indicate that an exception was detected (retriable or not, as indicated) and a REMU generated.

	States					
	NOT RCVD	IN TRANS	SUSP	TERM	COMPLETE	PURGED
Inputs	01	02	03	04	05	06
PREFIX RECEIVED(DTMU,DRMU)	2	-	/	-	/	-
PREFIX RECEIVED(DCMU)	/	-	2	-	/	-
DIST ACCEPTED	/	5	/	/	/	/
AFTER FAIL RESTART MID MU RESTART	-	3	-	-	-	-
AFTER FAIL RESTART NO MID MU RESTART	-	4	-	-	-	-
CONVERSATION FAILURE MID MU RESTART	/	3	/	/	/	/
CONVERSATION FAILURE NO MID MU RESTART	/	4	/	/	/	/
PROG ERROR RECEIVED MID MU RESTART	/	3	/	/	/	/
PROG ERROR RECEIVED NO MID MU RESTART	/	4	/	/	/	/
PRMU	/	/	6	6	6	-
SEMU-NO RETRY	4	-	4	-	-	-
SEMU-RETRY MID MU RESTART	4	-	-	-	-	-
SEMU-RETRY NO MID MU RESTART	4	-	/	-	-	-
REMU-NO RETRY	/	4	/	/	/	/
REMU-RETRY MID MU RESTART	/	3	/	/	/	/
REMU-RETRY NO MID MU RESTART	/	4	/	/	/	/

Chapter 3. Implementation Model

Introduction

This chapter presents an implementation model of a distribution service unit. The introductory section presents an overview of the transaction programs and data structures that comprise the DSU. The functions of the DSU are organized by DS sublayers. Diagrams illustrate the components of the DSU and the relationship of the DSU to the other parts of the network--the agents that use DS and the LU at which the DSU resides. The LU makes the services of the underlying communication network available to the DSU.

Following the introduction are groups of finite-state machines (FSMs) that illustrate the structure of the DSU in greater detail. The finite-state machines are grouped by DS sublayer. When the FSMs are executed, they generate sequences conforming to DS protocols.

The Structure of a DSU

Figure 38 on page 108 illustrates the components of the DSU and their relationship to the DS sublayers. (For simplification, the later diagrams do not show the DS sublayers.) The functions of the DS presentation services sublayer are provided by a DS presentation services program. The functions provided by the directing and routing sublayers are provided by the service transaction program DS_Router_Director. The functions of the distribution transport sublayer are provided by two service transaction programs, DS_Send and DS_Receive. An explanation of the numbered components of the DSU follows the figure.

The local-delivery queues are associated with both the presentation services and directing sublayers. The router-director queue is associated with both the directing and routing sublayers. The next-DSU queues are associated with both the routing and distribution transport sublayers.

The DS operations functions are associated with all DS sublayers. The programs that perform the operations functions for exception processing are invoked from the programs that detect the exception conditions.

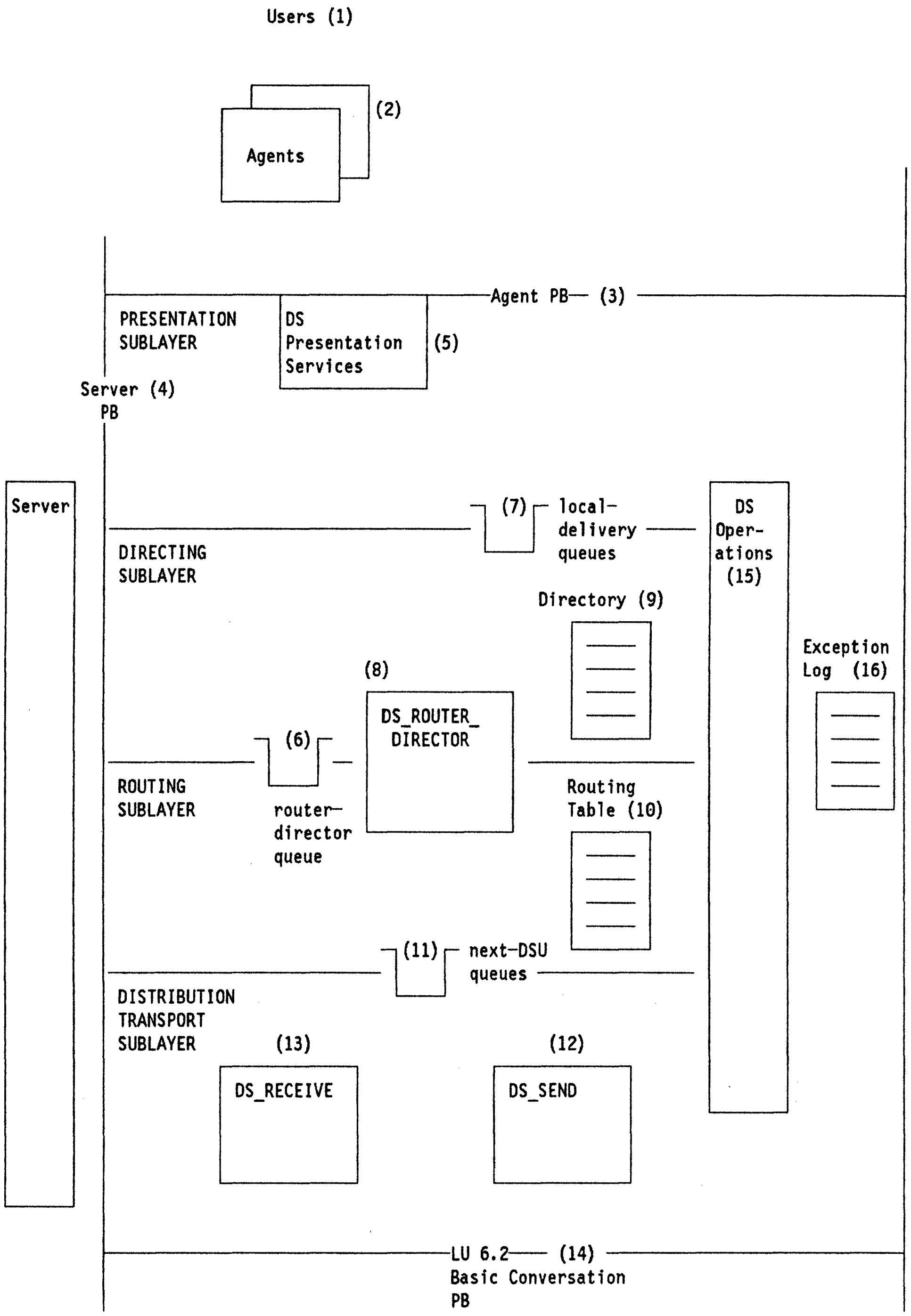


Figure 38. Structure of a DSU

The numbered components are:

1. *The users:* Users interact with transaction programs known as agents to send and receive distributions and to invoke operations functions.
2. *The agents:* The agents are above the DS protocol boundary. They interact with DS by issuing protocol boundary verbs. For outbound traffic, an agent issues a sequence of verbs known as a *sending sequence.* For distributions

to be delivered locally, DS selects the appropriate local-delivery queue according to contents of the directory and schedules the specified destination agent. The agent then issues a sequence of verbs known as a *receiving sequence* to receive the distribution. Further detail on the agent protocol boundary, sending sequences, and receiving sequences may be found in Chapter 1 and Appendix F.

3. *The agent protocol boundary:* It provides the interface between the agents and DS.

4. *The server protocol boundary:* It provides the interface between DS and the servers. For outbound distributions, DS gets read access to the server object by issuing verbs to the server across this boundary. For inbound distributions, DS issues verbs to store the server object.

5. *DS presentation services:* Presentation services validates the requests from the agents, schedules further distribution functions, and provides return codes and parameters to the agents.

6. *The router-director queue:* This queue contains distributions originated locally (placed on the queue by presentation services) or received from remote DSUs (placed on the queue by DS_Receive). When DS is required to generate distribution reports, DS operations places them on the queue. The queue is serviced by DS_Router_Director.

7. *The local-delivery queues:* These queues contain distributions awaiting delivery by DS presentation services to local agents. The directing function of DS_Router_Director places distributions on the queues; it selects the appropriate queue based on parameters in the directory and information in the distribution. Presentation services services the queue when an agent issues Receive_Distribution.

8. *DS_Router_Director:* This service transaction program performs the functions associated with both the routing and directing sublayers. It services the router-director queue, accesses the directory and the routing tables, and places distributions on the local-delivery queues or next-DSU queues.

9. *The directory:* It contains information that enables DS_Router_Director to associate a destination DSU name with a destination user and local delivery information with a local user.

10. *The routing table:* It contains information that enables DS_Router_Director to select the appropriate next-DSU queue for outbound distributions.

11. *The next-DSU queues:* These contain distributions to be transmitted to remote DSUs. DS_Router_Director places entries on the queues. DS_Send services the queues.

12. *DS_Send:* This service transaction program performs the functions required to send distributions to adjacent DSUs.

13. *DS_Receive:* This service transaction program performs the functions required to receive distributions from adjacent DSUs. It uses server verbs to store server objects and places distributions on the router-director queue.

14. *The LU 6.2 basic conversation protocol boundary:* This interface enables DS to issue LU 6.2 verbs to send and receive distributions.

15. *DS operations:* These programs perform functions required when exceptions are detected or when operations users display or change DSU information. These programs can be invoked from any of the other DS transaction programs.

16. *The exception log:* This log contains information concerning exception incidents detected by DS. DS creates the entries. DS operations programs access the entries for operators.

Examples of DSU Activity

The diagrams in this section show the interaction of the components described in "The Structure of a DSU" on page 107 for various distribution activities.

Origin of Distribution with Local Destinations

Figure 39 on page 111 shows the interaction of the DSU components when a user sends a distribution to another user or users, all local to the same DSU.

Refer to Figure 39 on page 111 for the following interactions.

1. The user interacts with an agent to initiate a distribution request.
2. The agent issues Send_Distribution to DS. Presentation services programs analyze the request information and accept it.
3. Presentation services signals SERVER_MGR to increment the DS lock count for the server object. SERVER_MGR issues Assign_Read_Access to the server.
4. The distribution is placed on the router-director queue and DS_Router_Director is scheduled.
5. An OK return code is returned to the agent. The agent issues Sending_Sequence_Completed.
6. DS_Router_Director accesses the distribution on the router-director queue. It recognizes that the distribution is at its origin and invokes directing.
7. Directing reads the directory and determines that all destination users are local.
8. Information in the directory is used to select the appropriate local-delivery queues for the distribution. The destination agent is scheduled. The identification of the queue is supplied to the agent.
9. The agent issues Receive_Distribution.
10. Presentation services accesses the named local-delivery queue and reads the distribution information.
11. The distribution information is returned to the agent. The agent issues Receiving_Sequence_Completed.
12. The agent provides the distribution information to the user.

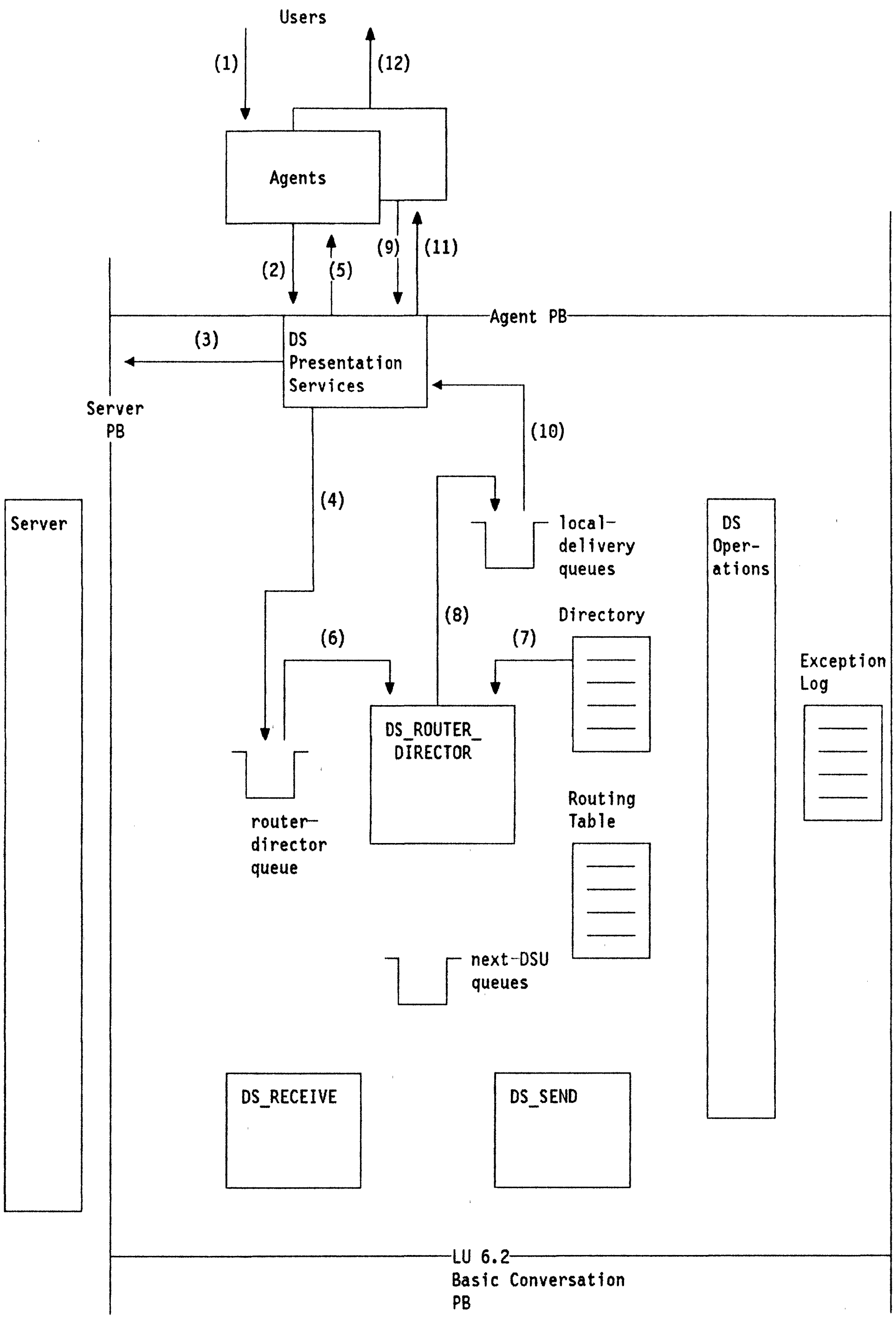


Figure 39. Processing at the Origin of a Distribution with Local Destinations

Origin of Distribution with Remote Destinations

Figure 40 shows the component interaction at the origin of a distribution that has remote destinations.

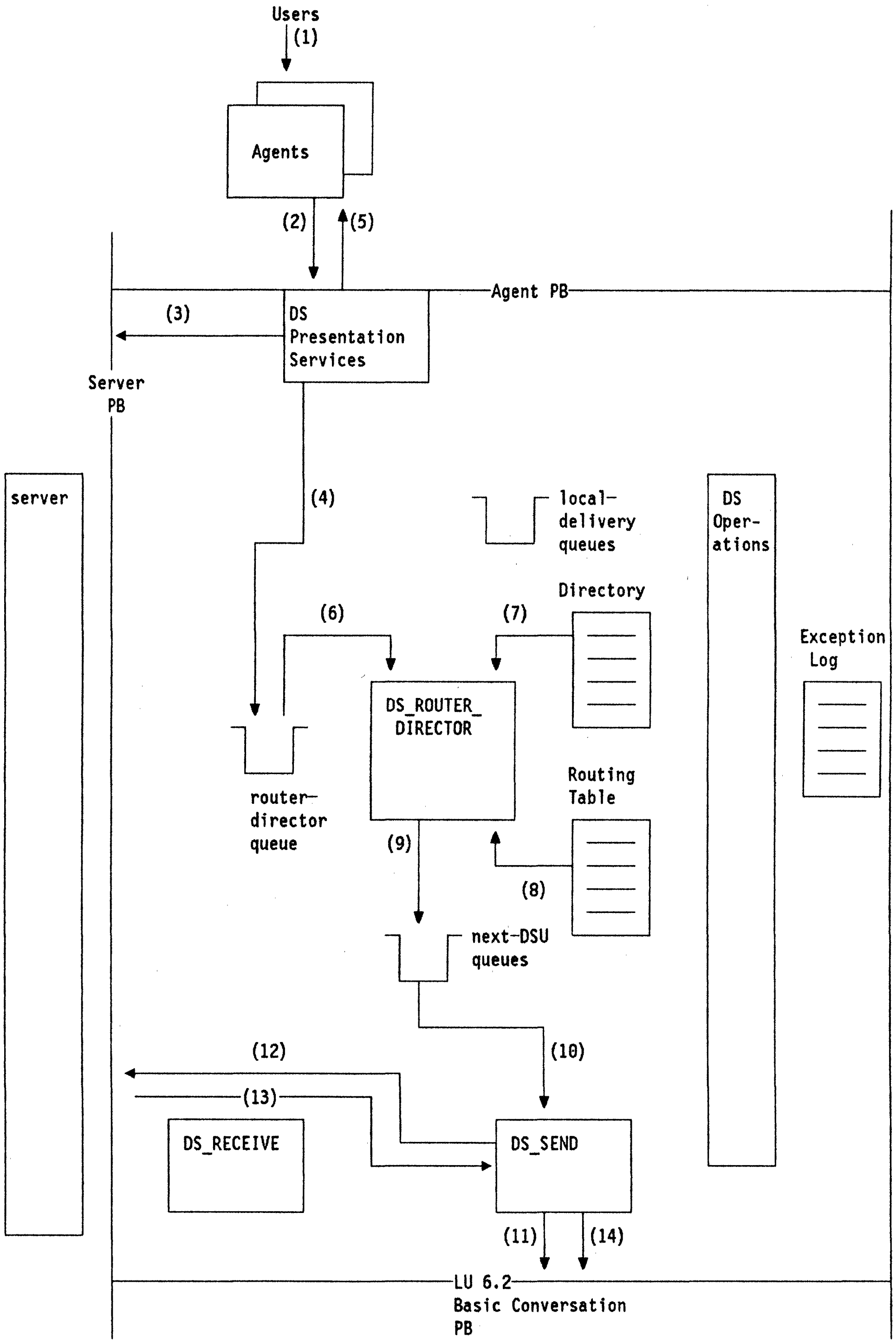


Figure 40. Processing at the Origin of a Distribution with Remote Destinations

1. The user interacts with an agent causing it to initiate a distribution request.
2. The agent issues Send_Distribution to DS. The request information is accepted by presentation services programs.
3. Presentation services signals the server manager to increment the DS lock count for the server object. The server manager issues Assign_Read_Access to the server.
4. The distribution is placed on the router-director queue, and DS_Router_Director is scheduled.
5. An OK return code is returned to the agent.
6. DS_Router_Director accesses the distribution on the router-director queue. It recognizes that the distribution is at its origin.
7. The directory contents are read to determine the destination DSU for each destination user. The distribution is passed to routing.
8. The routing table is accessed to determine the next-DSU queue for the distribution.
9. The information is placed on the selected next-DSU queue, and DS_Send is scheduled.
10. DS_Send accesses the next-DSU queue.
11. DS_Send starts a conversation via LU 6.2 with DS_Receive at the adjacent DSU. That DSU may or may not be the destination; the sending process is the same.
12. A server verb is issued to read the server object.
13. The server object is retrieved from the server and encoded as part of the DTMU. Multiple Read verbs may be required to read the entire server object.
14. The DTMU is transmitted to the adjacent DSU.

Destination of Distribution

Figure 41 illustrates the processing when a DSU receives a distribution whose destinations are local.

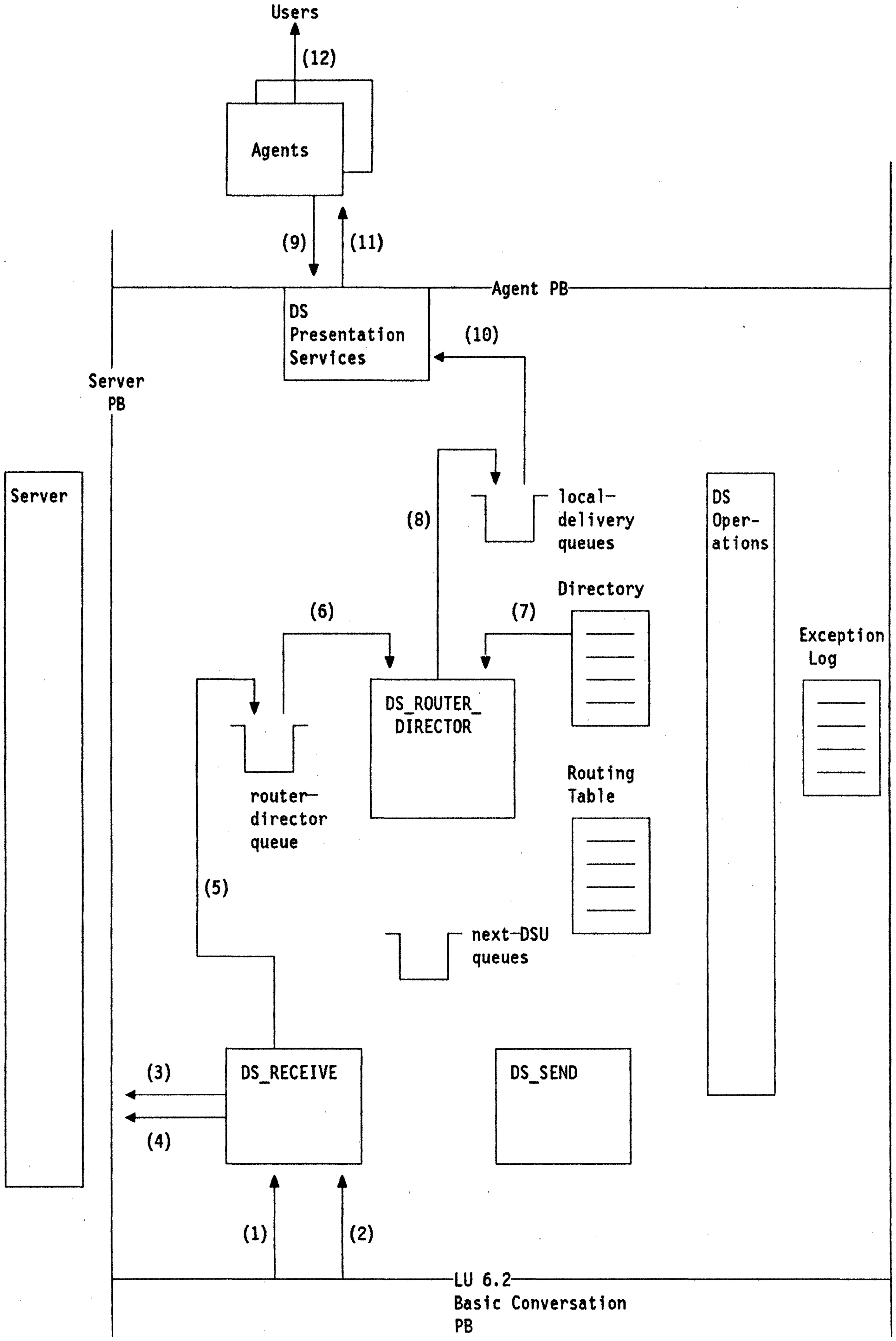


Figure 41. Processing a Received Distribution at the Destination

1. DS_Receive is attached by LU 6.2 as a result of activity initiated by DS_Send in the sending DSU (not shown).

2. The DTMU is received. Multiple LU 6.2 verbs may be issued to receive the entire DTMU.
3. DS_Receive issues server verbs to allocate space for the server object.
4. The server object is stored by the server.
5. The distribution is placed on the router-director queue and DS_Router_Director is scheduled.
6. DS_Router_Director accesses the distribution. Routing determines that the local DSU is a destination for the distribution and invokes directing.
7. Directing accesses the directory, determines that the distribution is to be delivered locally, and gets the local delivery information (local-delivery queue identifier and parameters).
8. Directing places the distribution on the appropriate local-delivery queue and schedules the destination agent. The queue identifier is passed to the agent.
9. The agent issues Receive_Distribution.
10. A presentation services program accesses the indicated local-delivery queue and reads the distribution.
11. The distribution information is returned to the agent.
12. The agent returns information to the user.

Processing a Received Distribution with a Routing Exception

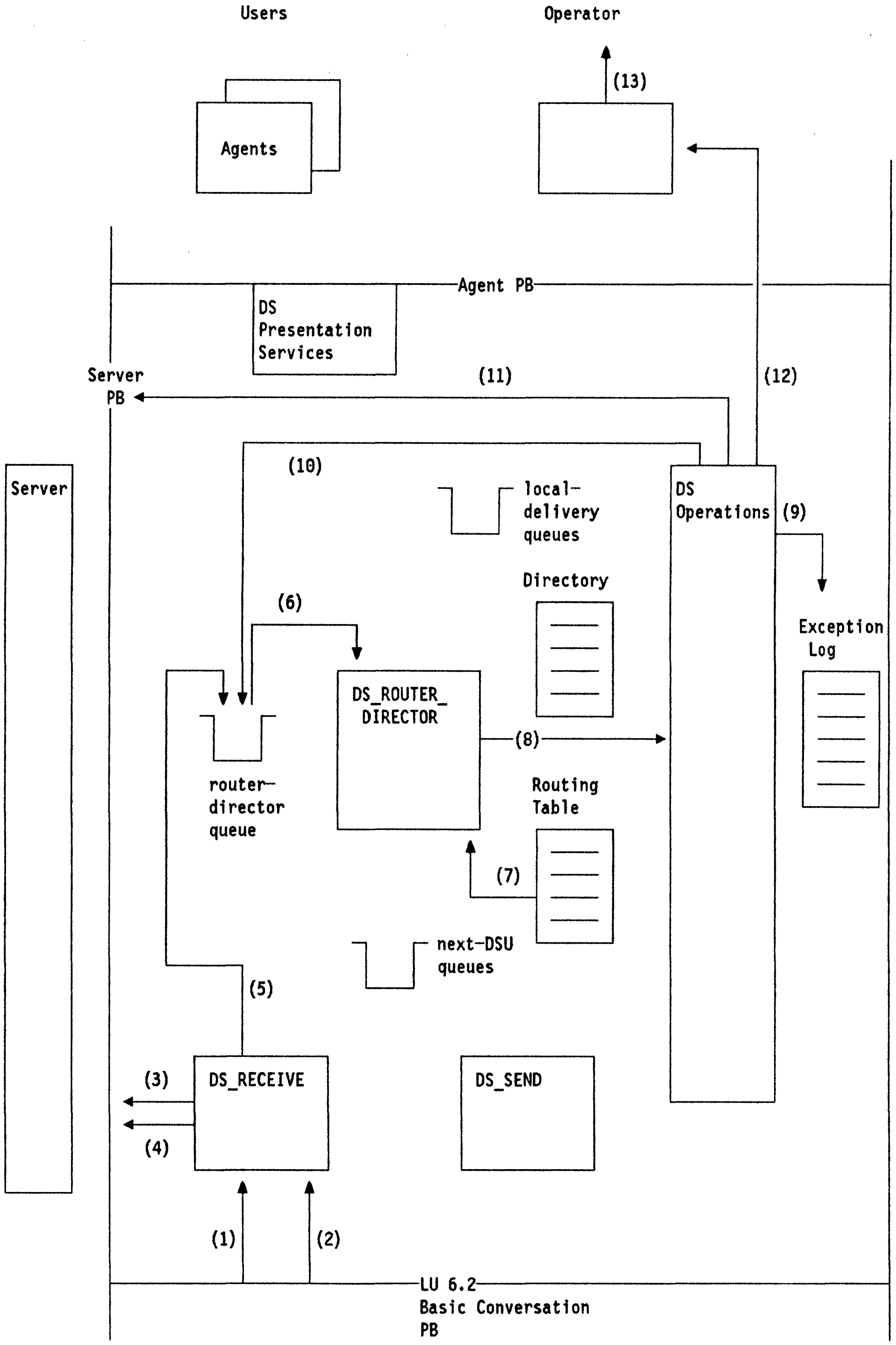


Figure 42. Processing a Received Distribution with a Routing Exception

Figure 42 on page 116 illustrates the processing performed by the DSU when a routing exception occurs during the processing of a distribution. For this illustration, assume that the routing table accessed by DS_Router_Director has an error so that the destination DSU is not listed in it. This could result from to an error in building the table or because the table has been improperly changed.

1. DS_Receive is attached by LU 6.2 as a result of activity (not shown) initiated by DS_Send in the sending DSU.
2. The DTMU is received. Multiple LU 6.2 verbs may be issued to receive the entire DTMU.
3. DS_Receive issues server verbs to allocate space for the server object.
4. The server object is stored by the server.
5. The distribution is placed on the router-director queue and DS_Router_Director is scheduled.
6. DS_Router_Director accesses the distribution. The destination DSU name in the distribution is not local.
7. DS_Router_Director accesses the routing table to select the next-DSU queue for the distribution. The destination DSU and service level required are not found in the routing table.
8. DS_Router_Director invokes DS Operations to process the exception condition.
9. DS Operations records exception information in the exception log.
10. DS Operations creates a distribution report, places it on the router-director queue, and schedules DS_Router_Director. From this point, the distribution report is processed the same as any other distribution on the router-director queue.
11. DS Operations releases the server object by issuing server verbs.
12. DS Operations issues a message to the operator indicating the exception.
13. An agent displays exception information to an operator.

Accessing Logged Exception

Figure 43 illustrates the processing when an operator accesses the exception information on the exception log.

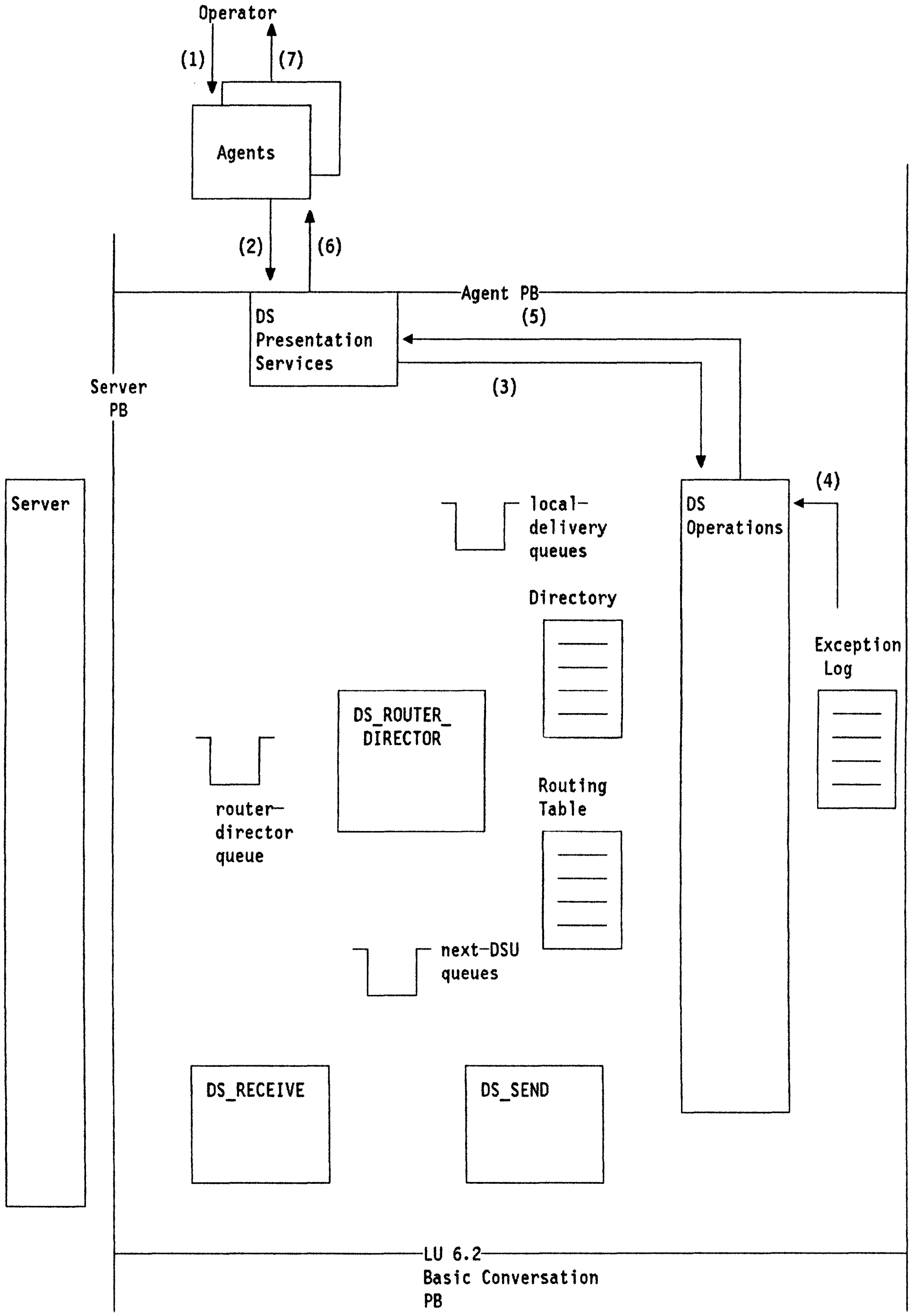


Figure 43. Accessing a Logged Exception

1. An operator interacts with an agent to access the contents of the exception log for a particular distribution.
2. The agent issues the Get_Exception_Log_Entry verb, identifying the distribution. A presentation services program accepts the verb.
3. The presentation services program invokes DS Operations to access the exception log with the identification of the required distribution.
4. DS Operations accesses the exception log and locates the distribution information.
5. DS Operations returns the information to the presentation services program.
6. The presentation services program returns the exception information to the requesting agent.
7. The agent formats the information for the operator.

Presentation Services Sublayer

This section presents a description of the functions performed by the presentation services sublayer in response to the various verbs that an agent may issue.

Send_Distribution

Presentation services performs the following actions in response to a Send_Distribution verb:

- The syntax of the verb and dependencies among the parameters are checked.
- The parameters of the verb are validated. For information on the required protocol boundary checking performed by all implementations, see Appendix C.
- The DS lock count for the server object is incremented. Assign_Read_Access is issued to the server, to ensure that DS has read access to the server object and that the server object will not be prematurely deleted.
- The distribution is placed on the router-director queue.
- DS_Router_Director is scheduled.
- The sending state for the distribution is updated. For a description of the possible values of the sending state, see Appendix F.
- Control is returned to the agent.

Query_Distribution_Sending

Presentation services performs the following actions in response to a Query_Distribution_Sending verb:

- The syntax of the verb and dependencies among the parameters are checked.
- The parameters of the verb are validated.

- The control block for the distribution is located and the distribution control information is obtained.
- Control is returned to the agent along with the appropriate distribution information.

Sending_Sequence_Completed

PS performs the following actions in response to a Sending_Sequence_Completed verb:

- The syntax of the verb and dependencies among the parameters are checked.
- The parameters of the verb are validated.
- The sending state is checked to see that the Sending_Sequence_Completed verb is appropriate.
- The distribution control block is located.
- The sending state for the distribution is updated as appropriate.
- Control is returned to the agent.

Receive_Distribution, Receive_Distribution_Report

PS performs the following actions in response to a Receive_Distribution or a Receive_Distribution_Report verb:

- The syntax of the verb and dependencies among the parameters are checked.
- The parameters of the verb are validated.
- The distribution is retrieved from the local delivery queue.
- The distribution control block is marked as received and a time stamp is saved.
- The distribution information is returned to the agent.

Receiving_Sequence_Completed

PS performs the following actions in response to a Receiving_Sequence_Completed verb:

- The syntax of the verb and dependencies among the parameters are checked.
- The parameters of the verb are validated.
- The distribution control block is located and checked to see that the distribution has previously been received.
- The agent lock count is decremented for the server object. If both the DS and agent lock counts are 0 following the decrement operation, Release_Read_Access is issued to the server to release access for both DS and the agent.
- The distribution copy is dequeued from the local delivery queue.

- If Receiving_Sequence_Completed has been issued for all copies of the distribution, the distribution control block is deleted.
- Control is returned to the agent.

Obtain_Local_Server_Report

PS performs the following actions in response to an Obtain_Local_Server_Report verb:

- The syntax of the verb and dependencies among the parameters are checked.
- The parameters of the verb are validated.
- The server report is obtained from the appropriate local delivery queue.
- The server report is returned to the agent.

Operations Verbs

For a description of the operations verbs, see Chapter 1 and Appendix F. PS performs the following actions in response to an operations verb:

- The syntax of the verb and dependencies among the parameters are checked.
- The parameters of the verb are validated.
- The requested operation is performed.
- The results of the verb are returned to the agent.

Routing and Directing Sublayers

Routing and Directing Overview

The formal model groups these functions together under a single transaction program, but DS implementations may choose other groupings. This model also shows some option subsets, which implementations may or may not decide to support. However, all DS implementations provide the base DS functions, as described in Appendix C.

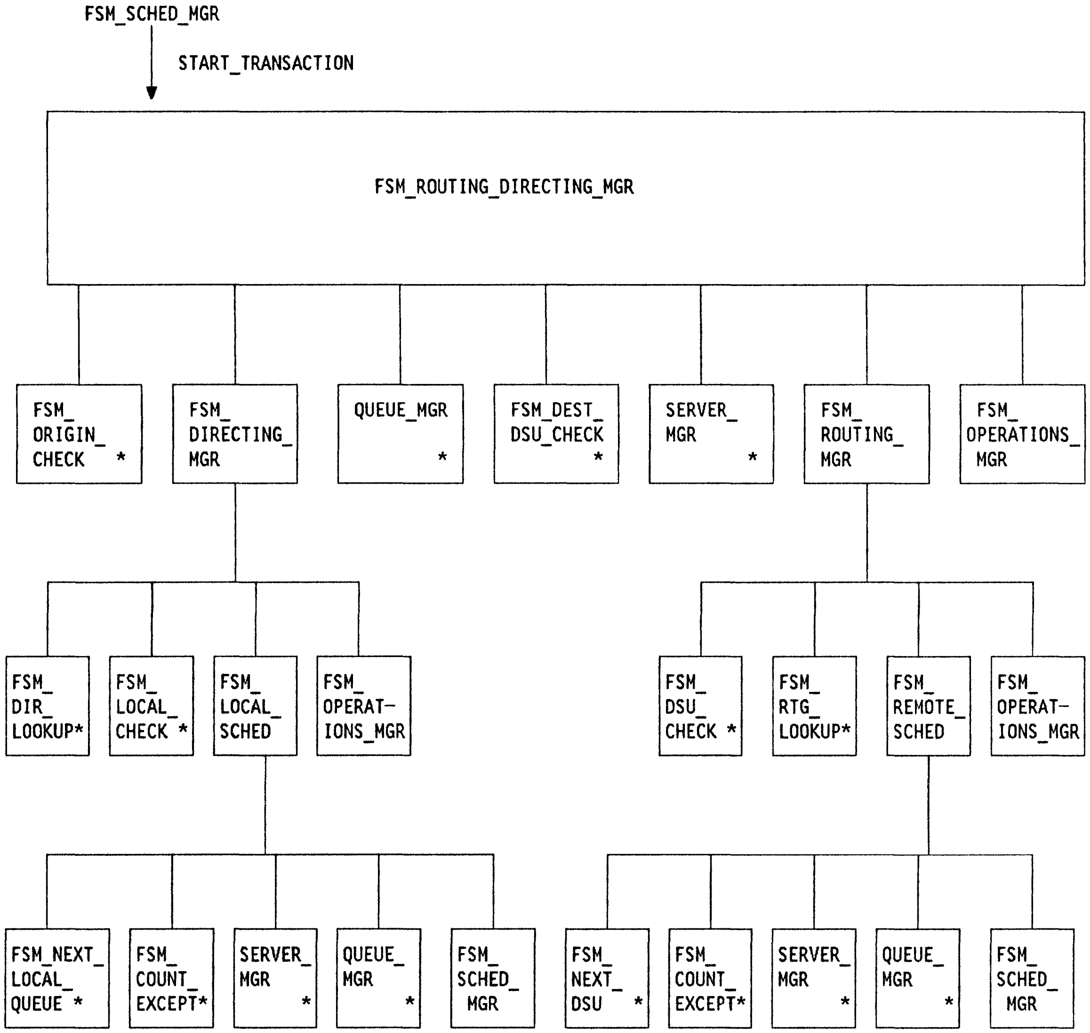
The machines in this section correspond to the transaction program DS_Router_Director. A manager machine controls lower-level machines in the hierarchy. The manager machine loops, removing distributions from the router-director queue, and signals lower-level machines to perform either the routing function or the directing function.

DS_Router_Director has the responsibility of enqueuing a distribution either locally for delivery, or on a next-DSU queue from which it will be transmitted to an adjacent DSU.

There are two components:

- *Routing* processes distributions originating locally or remotely, examines the DSU names, and either gives them to directing for local handling, or queues them again for transmission onward.

- *Directing* performs directory lookup and local delivery functions. For distributions that were originated locally, it supplies DSU names for user destinations that are to be transmitted for delivery elsewhere. Directing also delivers distributions locally; that is, it puts them on local queues and causes designated local agents to gain control. Another function of directing is that of *redirection.* In performing this function, the distribution is passed from routing to directing and back to routing with new destination DSUs for some user destinations. This communication takes place through the manager machine.



* indicates those finite-state machines not formally specified.

For more details regarding the finite-state machines, see:

- "FSM_ROUTING_DIRECTING_MGR" on page 124
- "FSM_LOCAL_SCHED" on page 132
- "FSM_DIRECTING_MGR" on page 128
- "FSM_REMOTE_SCHED" on page 140
- "FSM_ROUTING_MGR" on page 136
- "FSM_SCHED_MGR" on page 351
- "FSM_OPERATIONS_MGR" on page 342
- "FSM_ORIGIN_CHECK" on page 144
- "FSM_NEXT_LOCAL_QUEUE" on page 145
- "QUEUE_MGR" on page 357
- "FSM_COUNT_EXCEPTS" on page 145
- "FSM_DEST_DSU_CHECK" on page 144
- "FSM_DSU_CHECK" on page 145
- "SERVER_MGR" on page 354
- "FSM_RTG_LOOKUP" on page 145
- "FSM_DIR_LOOKUP" on page 144
- "FSM_NEXT_DSU" on page 146
- "FSM_LOCAL_CHECK" on page 144

Figure 44. DS_Router_Director FSM Hierarchy

FSM_ROUTING_DIRECTING_MGR

This machine controls the sequence of routing and directing functions for each distribution. It uses information in the distribution, and signals lower-level machines to accomplish its functions. In the case of exceptions reported by lower-level machines, this machine takes exception information passed from those machines and gives it to FSM_OPERATIONS_MGR for processing.

FSM_ROUTING_DIRECTING_MGR is started by a signal from FSM_SCHED_MGR. It processes entries on the router-director queue until the queue is empty. First, it signals QUEUE_MGR to remove a distribution from the router-director queue. When the distribution is dequeued, FSM_ROUTING_DIRECTING_MGR determines whether it originated locally and has at least one user destination. If it originated locally and at least one user destination is present, the distribution is passed to the directing manager, which performs local distribution and fills in DSUs for remote user destinations.

When FSM_ROUTING_DIRECTING_MGR receives the DIRECTING_COMPLETE signal, it passes the distribution to the routing manager, if distribution copies for remote destinations are to be processed. If, however, all the destinations happen to be local, it decrements the server-object DS lock count, which was incremented when the distribution entered the DSU, and asks for the next distribution on the router-director queue.

If the distribution originated at a remote DSU, FSM_ROUTING_DIRECTING_MGR passes the distribution to the routing manager first. If any of the destination DSUs are local, the routing manager requests that the directing function be performed on the distribution. If no DSUs are local, the routing manager puts the distributions on the appropriate next-DSU queues.

When FSM_ROUTING_DIRECTING_MGR receives the DIRECTING_COMPLETE signal, the distribution has been enqueued for delivery to any local users or agents, and these local destinations have been removed from the distribution. There may or may not be destinations remaining. When FSM_ROUTING_DIRECTING_MGR receives the ROUTING_COMPLETE signal, no more destinations are to be serviced for the distribution and DS_Router_Director processing is complete. Any fan-out is performed by the routing manager and the directing manager.

Only two exceptions are reported to FSM_ROUTING_DIRECTING_MGR: a server exception and a queue exception. In both these cases, it signals FSM_OPERATIONS_MGR to log the exception and send a message to the operator so that the affected resources can be repaired.

Exceptions in directing and routing are handled by FSM_DIRECTING_MGR and FSM_ROUTING_MGR. The DIRECTING_COMPLETE and the ROUTING_COMPLETE signals imply that any exceptions have been processed and that erroneous user names and DSUs, or destinations for which delivery could not be made, have been erased from the distribution.

Function: This finite-state machine describes the sequence of routing and directing functions in DS_Router_Director. For a further description, see "FSM_ROUTING_DIRECTING_MGR" on page 124.

This FSM gets control from one of the following:

- Signals from finite-state machines providing common services:
  - START_TRANSACTION from FSM_SCHED_MGR
  - QUEUE_OK from QUEUE_MGR
  - QUEUE_NOT_OK from QUEUE_MGR
  - QUEUE_EMPTY from QUEUE_MGR
  - OPERATIONS_COMPLETE from FSM_OPERATIONS_MGR
  - OBJECT_OK from SERVER_MGR
  - OBJECT_NOT_OK from SERVER_MGR
- Signals from lower-level routing and directing finite-state machines:
  - ORIGIN_REQUEST_USER_NAME from FSM_ORIGIN_CHECK
  - ORIGIN_REQUEST_NO_USER_NAME from FSM_ORIGIN_CHECK
  - NOT_ORIGIN_REQUEST from FSM_ORIGIN_CHECK
  - DIRECTING_COMPLETE from FSM_DIRECTING_MGR
  - ROUTING_COMPLETE from FSM_ROUTING_MGR
  - ROUTING_RESET from FSM_ROUTING_MGR
  - DIRECTING_REQUIRED from FSM_ROUTING_MGR
  - DEST_DSUS_REMAINING from FSM_DEST_DSU_CHECK
  - NO_DEST_DSUS_REMAINING from FSM_DEST_DSU_CHECK

	States										
	RESET	READ QUE PEND	ORIG CHK	DIR	RTG	DSUS PEND	DEC LOCK	DEQ PEND	ERP	OPER PEND	RESET RTG
Inputs	01	02	03	04	05	06	07	08	09	10	11
START TRANSACTION	2a	/	/	/	/	/	/	/	/	/	/
QUEUE OK	/	3b	/	/	/	/	/	2a	/	/	/
QUEUE NOT OK	/	9f	/	/	/	/	/	9f	/	/	/
QUEUE EMPTY	/	1	/	/	/	/	/	/	/	/	/
ORIGIN REQUEST USER NAME	/	/	4c	/	/	/	/	/	/	/	/
ORIGIN REQUEST NO USER NAME	/	/	5d	/	/	/	/	/	/	/	/
NOT ORIGIN REQUEST	/	/	5d	/	/	/	/	/	/	/	/
DIRECTING COMPLETE	/	/	/	6g	/	/	/	/	/	/	/
ROUTING COMPLETE	/	/	/	/	7e	/	/	/	/	/	/
DIRECTING REQUIRED	/	/	/	/	4c	/	/	/	/	/	/
ROUTING RESET	/	/	/	/	/	/	/	/	/	/	7e
DEST DSUS REMAINING	/	/	/	/	/	5d	/	/	/	/	/
NO DEST DSUS REMAINING	/	/	/	/	/	11i	/	/	/	/	/
OBJECT OK	/	/	/	/	/	/	8h	/	/	/	/
OBJECT NOT OK	/	/	/	/	/	/	10f	/	/	/	/
OPERATIONS COMPLETE	/	/	/	/	/	/	/	/	2a	8h	/

Output Code	Function
a	Signal QUEUE_MGR with READQ to read the next available entry from the router-director queue.
b	Signal FSM_ORIGIN_CHECK with CHECK_FOR_ORIGIN to determine if the distribution is at the origin.
c	Signal FSM_DIRECTING_MGR with DIRECT to determine the DSU for each destination.
d	Signal FSM_ROUTING_MGR with ROUTE to determine local and remote destinations, perform fan-out, and perform scheduling for remote destinations, if all destinations are remote.
e	Signal SERVER_MGR with DECREMENT_OBJ_LOCK to decrement the DS lock count for the server object. If no server object exists, SERVER_MGR returns OBJECT_OK.
f	Signal FSM_OPERATIONS_MGR with LOG_MESSAGE_EXCEPT to perform logging and to display an operator message. Any asynchronous reporting required was generated from either directing or routing.
g	Signal FSM_DEST_DSU_CHECK with CHECK_DSUS to determine if any destination DSUs remain to be handled by routing.
h	Signal QUEUE_MGR with DEQ to remove the entry processed from the Router-Director queue.
i	Signal FSM_ROUTING_MGR with ALL_LOCAL to indicate no more DSUs remain to be processed.

Directing FSMs

FSM_DIRECTING_MGR

The directing manager accesses the user directory and handles the local-delivery and redirection functions. The directing manager:

- Looks up user names and agents in the directory and obtains DSUs and local-delivery queue information where necessary.
- Signals the DS operations manager to handle any exceptions found in the destination list.
- Checks whether any local recipients are in the destination list.
- Puts the distribution on local-delivery queues for local recipients and removes them from the destination list.
- Passes any redirected destinations back to the FSM_ROUTING_DIRECTING_MGR.

The directing manager receives one of two types of input from the FSM_ROUTING_DIRECTING_MGR:

- A distribution that originated locally.
- A distribution that originated at a remote DSU, and that the routing manager has determined to have at least one destination local to this DSU. This type of distribution is assumed to have the apparent local destination DSUs and user names marked for the directing manager.

For destination users and agents, the directing manager first signals FSM_DIR_LOOKUP to access the user directory. Next, the directing manager signals FSM_LOCAL_CHECK to check the queue information appended by FSM_DIR_LOOKUP to determine if the distribution can be delivered locally. FSM_LOCAL_CHECK will fan out the distribution if necessary.

If no queue information was appended, then no local deliveries are to be made, and directing signals the FSM_ROUTING_DIRECTING_MGR that the directing function is complete. If local deliveries are to be made, the directing manager machine signals FSM_LOCAL_SCHED to queue the distribution to each of the local recipients in the destination list. When FSM_LOCAL_SCHED is done, the directing manager signals back to the FSM_ROUTING_DIRECTING_MGR that directing is done.

If exceptions are found in the directing operation, the directing manager machine signals the operations manager to handle the exceptions. Operations is given a list of the exception destinations, logs the exception, builds the distribution exception report, and notifies the operator if necessary.

Upon signalling back to FSM_ROUTING_DIRECTING_MGR, the directing manager has pruned all local recipients from the destination list, so that either no recipient remains, or only recipients at remote DSUs remain. Either all local recipients have had the distribution enqueued, or, if there were exceptions, the proper exception recovery action was taken.

Function: This finite-state machine describes the functional processing in directing. For a further description, see "FSM_DIRECTING_MGR" on page 128.

This FSM gets control from one of the following:

- Signals from higher-level routing and directing finite-state machines:
  - DIRECT from FSM_ROUTING_DIRECTING_MGR
- Signals from lower-level routing and directing finite-state machines:
  - DIR_LOOKUP_COMPLETE_NO_EXPTS from FSM_DIR_LOOKUP
  - DIR_LOOKUP_COMPLETE_EXPTS from FSM_DIR_LOOKUP
  - DIRECTING_LOCALS from FSM_LOCAL_CHECK
  - DIRECTING_NO_LOCALS from FSM_LOCAL_CHECK
  - DIRECTING_LOCALS_COPY_EXPT from FSM_LOCAL_CHECK
  - LOCAL_SCHED_COMPLETE from FSM_LOCAL_SCHED
  - LOCAL_SCHED_COMPLETE_EXPTS from FSM_LOCAL_SCHED
- Signals from finite-state machines providing common services:
  - OPERATIONS_COMPLETE from FSM_OPERATIONS_MGR

	States						
	RESET	DIR LOOKUP	DIR EXPT	LCL PEND	COPY ERP	LCL SCHED	SCHED ERP
Inputs	01	02	03	04	05	06	07
DIRECT	2a	/	/	/	/	/	/
DIR LOOKUP COMPLETE NO EXPTS	/	4b	/	/	/	/	/
DIR LOOKUP COMPLETE EXPTS	/	3c	/	/	/	/	/
DIRECTING LOCALS	/	/	/	6e	/	/	/
DIRECTING NO LOCALS	/	/	/	1d	/	/	/
DIRECTING LOCALS COPY EXPT	/	/	/	5c	/	/	/
LOCAL SCHED COMPLETE	/	/	/	/	/	1d	/
LOCAL SCHED COMPLETE EXPTS	/	/	/	/	/	7c	/
OPERATIONS COMPLETE	/	/	4b	/	1d	/	1d

Output Code	Function
a	Signal FSM_DIR_LOOKUP with FIND_DSU to determine the DSU for the user names that are not local, and local queue information for local user names and agents.
b	Signal FSM_LOCAL_CHECK with CHECK_LOCALS to determine if local queue information exists for any destinations. If local destinations exist, FSM_LOCAL_CHECK makes a copy of the server object, using the appropriate server.
c	Signal FSM_OPERATIONS_MGR with ROUTING_DIRECTING_EXCEPT to notify the operator, generate any requested reports, and to log.
d	Signal FSM_ROUTING_DIRECTING_MGR with DIRECTING_COMPLETE to indicate that all destination users have been completed with DSUs, any local distribution has been performed, and any exceptions have been processed by FSM_OPERATIONS_MGR.
e	Signal FSM_LOCAL_SCHED with LOCAL_DISTRIBUTIONS to schedule the destination agent for processing of local distributions.

FSM_LOCAL_SCHED

This machine shows the sequence of functions comprising the local delivery mechanism in DS. FSM_LOCAL_SCHED uses the local-delivery queue information and the copies of the distribution that FSM_LOCAL_CHECK made.

The directing manager machine signals FSM_LOCAL_SCHED when it determines that it has local destinations for the distribution. For each local-delivery queue identifier, FSM_LOCAL_SCHED does the following:

- Requests the next queue identifier and a copy of the distribution from FSM_NEXT_LOCAL_QUEUE.
- Increments the server object agent lock count.
- Enqueues the copy of the distribution on the local-delivery queue specified by the queue identifier.
- Schedules the destination agent identified in the distribution.
- Makes the queue entry available to the agent by a RELEASEQ signal to QUEUE_MGR.

If an exception occurs during the processing, FSM_LOCAL_SCHED does the following:

- Reverses any operation that has been done on the distribution up to that point.

  For example, if the server object agent lock count has been incremented and the distribution put on the local-delivery queue, FSM_LOCAL_SCHED dequeues the distribution and decrements the server-object agent lock count.
- Adds the destinations in the destination list of the reported-on distribution to a general exception list to be passed back to the directing manager machine.

When all the local-delivery queue identifiers are exhausted, FSM_LOCAL_SCHED checks for any exceptions that have occurred in the process, and returns the list of destinations for which the distribution could not be enqueued. If no exceptions were found, FSM_LOCAL_SCHED reports LOCAL_SCHED_COMPLETE back to the directing manager machine.

Although it is not shown in this machine, for an unsuccessful server, queue, or scheduling function, FSM_OPERATIONS_MGR should also be signalled with MESSAGE_TO_OPERATOR and passed the message that the operation could not be completed, and that, in handling the exception, another exception occurred.

Function: This finite-state machine describes the functional processing for local delivery in directing. For a further description, see "FSM_LOCAL_SCHED" on page 132.

This FSM gets control from one of the following:

- Signals from higher-level routing and directing finite-state machines:
  - LOCAL_DISTRIBUTIONS from FSM_DIRECTING_MGR
- Signals from lower-level routing and directing finite-state machines:
  - NEXT_QUEUE_ID from FSM_NEXT_LOCAL_QUEUE
  - END_QUEUE_IDS from FSM_NEXT_LOCAL_QUEUE
  - SCHED_NO_EXCEPTS from FSM_COUNT_EXCEPTS
  - SCHED_EXCEPTS from FSM_COUNT_EXCEPTS
- Signals from finite-state machines providing common services:
  - OBJECT_OK from SERVER_MGR
  - OBJECT_NOT_OK from SERVER_MGR
  - QUEUE_OK from QUEUE_MGR
  - QUEUE_NOT_OK from QUEUE_MGR
  - SCHED_FUNCTION_OK from FSM_SCHED_MGR
  - SCHED_FUNCTION_NOT_OK from FSM_SCHED_MGR

	States							
	RESET	**LOCAL QUEUE ENTRY**	**LOCK PEND**	**ENQ PEND**	**SCHED PEND**	**RELQ PEND**	**ERP**	**EXPT CNT**
Inputs	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**
LOCAL DISTRIBUTIONS	2a	/	/	/	/	/	/	/
NEXT QUEUE ID	/	3b	/	/	/	/	/	/
END QUEUE IDS	/	8h	/	/	/	/	/	/
OBJECT OK	/	/	4c	/	/	/	2e	/
OBJECT NOT OK	/	/	2e	/	/	/	2e	/
QUEUE OK	/	/	/	5f	/	2a	-d	/
QUEUE NOT OK	/	/	/	7d	/	7d	-d	/
SCHED FUNCTION OK	/	/	/	/	6k	/	/	/
SCHED FUNCTION NOT OK	/	/	/	/	7g	/	/	/
SCHED NO EXCEPTIONS	/	/	/	/	/	/	/	1i
SCHED EXCEPTIONS	/	/	/	/	/	/	/	1j

Output Code	Function
a	Signal FSM_NEXT_LOCAL_QUEUE with FIND_NEXT_QUEUE_ID to determine the next local-delivery queue for which there is a distribution.
b	Signal SERVER_MGR with INCREMENT_OBJ_LOCK to increment the agent lock count for the server object. If no server object exists, SERVER_MGR returns OBJECT_OK.
c	Signal QUEUE_MGR with WRITEQ to place the distribution on the local-delivery queue.
d	Signal SERVER_MGR with DECREMENT_OBJ_LOCK to decrement the agent lock count for the server object. If no server objects exists, SERVER_MGR returns OBJECT_OK.
e	Add exception to exception-user information. Signal FSM_NEXT_LOCAL_QUEUE with FIND_NEXT_QUEUE_ID to determine the next local-delivery queue for which there is a distribution.
f	Signal FSM_SCHED_MGR with START_REQUEST to schedule the destination agent.
g	Signal QUEUE_MGR with DEQ to remove the distribution from the local-delivery queue.
h	Signal FSM_COUNT_EXCEPTS with COUNT_SCHED_EXCEPTS to determine if exceptions existed in any of the local scheduling process.
i	Signal FSM_DIRECTING_MGR with LOCAL_SCHED_COMPLETE to signal directing that the local scheduling process was completed successfully.
j	Signal FSM_DIRECTING_MGR with LOCAL_SCHED_COMPLETE_EXPTS to signal directing that exceptions existed in the local scheduling process.
k	Signal QUEUE_MGR with RELEASEQ to remove the in-use mark from the queue entry in the local-delivery queue. The entry will then be available to the destination agent for processing.

Routing FSMs

FSM_ROUTING_MGR

The routing manager sequences the functions necessary to put a distribution on one or more next-DSU queues for transmission to adjacent DSUs. The input to this machine is the destination list. The output depends on whether there are any local DSUs in the destination list.

The routing manager generates two main processing sequences:

- When the routing manager determines that at least one local DSU is in the destination list, it signals DIRECTING_REQUIRED to FSM_ROUTING_DIRECTING_MGR. The output is the distribution with all the local destinations marked in the destination list.
- When the routing manager determines that no local DSUs are in the destination list, it sequences the routing table lookup, fan-out functions, and the enqueuing and scheduling functions. In this case, all copies of the distribution are put on the appropriate next-DSU queues.

The routing manager is aware of two kinds of exceptions: *routing table lookup exceptions* and *scheduling exceptions*. If routing table lookup exception conditions exist for all the destination DSUs, the routing manager machine signals the operations manager to process the exceptions and terminates processing. If exception conditions exist for only some of the DSUs, the routing manager handles the exceptions by a signal to the operations manager, and then tries to enqueue and schedule the distribution for the non-exception destinations. In the case of exceptions reported by FSM_REMOTE_SCHED, the routing manager passes the exception list to the operations manager. It assumes that all the non-exception copies of the distribution have been enqueued and scheduled.

Function: This finite-state machine describes the functional processing in routing. For a further description, see "FSM_ROUTING_MGR" on page 136.

This FSM gets control from one of the following:

- Signals from higher-level routing and directing finite-state machines:
  - ALL_LOCAL from FSM_ROUTING_DIRECTING_MGR
  - ROUTE from FSM_ROUTING_DIRECTING_MGR
- Signals from lower-level routing and directing finite-state machines:
  - ALL_REMOTE from FSM_DSU_CHECK
  - AT_LEAST_ONE_LOCAL from FSM_DSU_CHECK
  - RTG_LOOKUP_COMPLETE from FSM_RTG_LOOKUP
  - RTG_LOOKUP_COMPLETE_EXPTS from FSM_RTG_LOOKUP
  - RTG_LOOKUP_COMPLETE_ALL_EXPTS from FSM_RTG_LOOKUP
  - REMOTE_SCHED_COMPLETE from FSM_REMOTE_SCHED
  - REMOTE_SCHED_COMPLETE_EXPTS from FSM_REMOTE_SCHED
- Signals from finite-state machines providing common services:
  - OPERATIONS_COMPLETE from FSM_OPERATIONS_MGR

	States						
	RESET	DSU CHK	NEXT DSU	RTG EXPT	RMT SCHED	ERP	DIR PEND
Inputs	01	02	03	04	05	06	07
ROUTE	2a	/	/	/	/	/	3b
ALL LOCAL	-g	/	/	/	/	/	1g
ALL REMOTE	/	3b	/	/	/	/	/
AT LEAST ONE LOCAL	/	7c	/	/	/	/	/
RTG LOOKUP COMPLETE	/	/	5d	/	/	/	/
RTG LOOKUP COMPLETE EXPTS	/	/	4e	/	/	/	/
RTG LOOKUP COMPLETE ALL EXPTS	/	/	6e	/	/	/	/
REMOTE SCHED COMPLETE	/	/	/	/	1f	/	/
REMOTE SCHED COMPLETE EXPTS	/	/	/	/	6e	/	/
OPERATIONS COMPLETE	/	/	/	5d	/	1f	/

Output Code	Function
a	Signal FSM_DSU_CHECK with CHECK_LOCAL_REMOTE to determine if both local and remote destinations exist in the distribution.
b	Signal FSM_RTG_LOOKUP with FIND_NEXT_DSU to determine the next DSU connection information for the destination DSUs in the distribution. Hop counts are set if the DSU is the origin, or decremented and validated if the DSU is not the origin.
c	Signal FSM_ROUTING_DIRECTING_MGR with DIRECTING_REQUIRED to indicate that destinations local to the DSU exist, and require local delivery or redirection.
d	Signal FSM_REMOTE_SCHED with REMOTE_DISTRIBUTIONS to schedule the sending of distributions to next DSUs.
e	Signal FSM_OPERATIONS_MGR with ROUTING_DIRECTING_EXCEPT to notify the operator, generate any requested reports, and log.
f	Signal FSM_ROUTING_DIRECTING_MGR with ROUTING_COMPLETE to indicate that all remote destinations in the distribution have been fanned-out and the resulting distributions have been scheduled for sending to the next DSUs.
g	Signal FSM_ROUTING_DIRECTING_MGR with ROUTING_RESET to indicate that routing has been reset.

FSM_REMOTE_SCHED

This machine performs the sequence of functions necessary to enqueue a distribution for transmission to a remote destination. Its input is the distribution, and, in particular, the destination list. Its output is a list of DSUs that could not be sent, and copies of the distribution put on the correct next-DSU queues. It causes DS_Send to gain control to transmit the queued distributions.

For each next-DSU queue name in the list associated with the distribution, FSM_REMOTE_SCHED does the following:

- Requests a next-DSU queue name and a copy of the distribution from FSM_NEXT_DSU.
- Requests that the server manager increment the DS lock count for the server object. This creates a lock on the server object for each copy of the distribution that must be transmitted.
- Requests that the queue manager put a copy of the server object on the named next-DSU queue.
- Requests that the scheduler start DS_Send, as appropriate to the scheduling algorithms.

If an exception occurs, the operations that have been performed on the distribution up to that point are reversed. FSM_REMOTE_SCHED adds the exception DSUs to a list that it passes back to the routing manager machine. In general, if, during the exception recovery process, another exception occurs, it is noted, but no other action is taken.

When FSM_REMOTE_SCHED has no more next-DSU queues to which to route the distribution, it signals FSM_COUNT_EXCEPTS to check whether the exception list is empty.

If it is not empty, FSM_REMOTE_SCHED signals the routing manager machine with REMOTE_SCHED_COMPLETE_EXCEPTS so that the routing manager can signal operations. Otherwise, FSM_REMOTE_SCHED signals REMOTE_SCHED_COMPLETE.

Although it is not shown in this machine, for an unsuccessful server, queue, or scheduling function, FSM_OPERATIONS_MGR should also receive a MESSAGE_TO_OPERATOR signal and the message that the function could not be completed and that in handling the exception, another exception occurred.

Function: This finite-state machine describes the functional processing for remote destinations in routing. For a further description, see "FSM_REMOTE_SCHED" on page 140.

This FSM gets control from one of the following:

- Signals from higher-level routing and directing finite-state machines:
  - REMOTE_DISTRIBUTIONS from FSM_ROUTING_MGR
- Signals from lower-level routing and directing finite-state machines:
  - NEXT_DSU from FSM_NEXT_DSU
  - END_NEXT_DSUS from FSM_NEXT_DSU
  - SCHED_NO_EXCEPTS from FSM_COUNT_EXCEPTS
  - SCHED_EXCEPTS from FSM_COUNT_EXCEPTS
- Signals from finite-state machines providing common services:
  - OBJECT_OK from SERVER_MGR
  - OBJECT_NOT_OK from SERVER_MGR
  - QUEUE_OK from QUEUE_MGR
  - QUEUE_NOT_OK from QUEUE_MGR
  - SCHED_FUNCTION_OK from FSM_SCHED_MGR
  - SCHED_FUNCTION_NOT_OK from FSM_SCHED_MGR

	States							
	RESET	NEXT DSU ENTRY	LOCK PEND	WRITEQ PEND	SCHED PEND	RELQ PEND	ERP	EXPT CNT
Inputs	01	02	03	04	05	06	07	08
REMOTE DISTRIBUTIONS	2a	/	/	/	/	/	/	/
NEXT DSU	/	3b	/	/	/	/	/	/
END NEXT DSUS	/	8h	/	/	/	/	/	/
OBJECT OK	/	/	4c	/	/	/	2e	/
OBJECT NOT OK	/	/	2e	/	/	/	2e	/
QUEUE OK	/	/	/	5f	/	2a	-d	/
QUEUE NOT OK	/	/	/	7d	/	7d	-d	/
SCHED FUNCTION OK	/	/	/	/	6k	/	/	/
SCHED FUNCTION NOT OK	/	/	/	/	7g	/	/	/
SCHED NO EXCEPTIONS	/	/	/	/	/	/	/	1i
SCHED EXCEPTIONS	/	/	/	/	/	/	/	1j

Output Code	Function
a	Signal FSM_NEXT_DSU with FIND_NEXT_DSU to determine the next DSU for which there is a distribution.
b	Signal SERVER_MGR with INCREMENT_OBJ_LOCK to increment the DS lock count for the server object. If no server object exists, SERVER_MGR returns OBJECT_OK.
c	Signal QUEUE_MGR with WRITEQ to place the distribution on the next-DSU queue.
d	Signal SERVER_MGR with DECREMENT_OBJ_LOCK to decrement the DS lock count for the server object. If no server object exists, SERVER_MGR returns OBJECT_OK.
e	Add exception to exception-user information. Signal FSM_NEXT_DSU with FIND_NEXT_DSU to determine the next DSU for which there is a distribution.
f	Signal FSM_SCHED_MGR with START_REQUEST to schedule the DS_Send transaction program to send distributions to the next DSU.
g	Signal QUEUE_MGR with DEQ to remove the distribution from the next-DSU queue.
h	Signal FSM_COUNT_EXCEPTS with COUNT_SCHED_EXCEPTS to determine if exceptions occurred in any part of the remote scheduling process.
i	Signal FSM_ROUTING_MGR with REMOTE_SCHED_COMPLETE to indicate to routing that the remote scheduling process was completed successfully.
j	Signal FSM_ROUTING_MGR with REMOTE_SCHED_COMPLETE_EXPTS to indicate to routing that exceptions occurred in the remote scheduling process.
k	Signal QUEUE_MGR with RELEASEQ to remove the in-use mark from the queue entry in the next-DSU queue. The entry is then available to DS_Send for processing.

Routing and Directing Utility FSMs

FSM_ORIGIN_CHECK

This FSM determines whether the distribution passed to it originated locally or at a remote DSU, and whether user destinations are present. FSM_ORIGIN_CHECK reports the results of this operation to FSM_ROUTING_DIRECTING_MGR by signalling either ORIGIN_REQUEST_USER_NAMES if the distribution originated at the local DSU and if at least one user name is in the destination list. It returns ORIGIN_REQUEST_NO_USER_NAMES if the distribution originated at the local DSU, but the destination list includes only node destinations. It returns NOT_ORIGIN_REQUEST otherwise. FSM_ROUTING_DIRECTING_MGR decides on the basis of these signals whether to signal the directing manager or the routing manager first.

FSM_DEST_DSU_CHECK

This FSM determines whether any destinations remain in the distribution to be serviced. FSM_ROUTING_DIRECTING_MGR manager signals this machine after the directing manager is finished in order to decide whether to signal the routing manager. If the destination list is empty, the directing manager has delivered the distribution locally, and no recipients remain. In this case, it signals NO_DEST_DSUS_REMAINING back to FSM_ROUTING_DIRECTING_MGR. If the list is not empty, then there are recipients at remote DSUs to which the distribution must be routed. In this case, it signals DEST_DSUS_REMAINING back to FSM_ROUTING_DIRECTING_MGR.

FSM_DIR_LOOKUP

If the distribution originated locally, the directory lookup produces DSU names for all user names that are not local, and local-delivery queue information for the user names that are local. For a distribution that originated at another DSU, the directory lookup returns a new DSU name if redirection has taken place, or local-delivery queue information if the user name is truly local.

If the directory lookup produces an exception (for example, ENTRY_NOT_FOUND), then the user name producing the exception is recorded. When all the user directory accesses are done, the user names producing exceptions and the valid user names are returned to the directing manager machine with the signal DIR_LOOKUP_COMPLETE_EXPTS.

If no exceptions are encountered, the destination list is returned with the signal DIR_LOOKUP_COMPLETE_NO_EXPTS.

FSM_LOCAL_CHECK

This machine determines from the destination list whether local-delivery queue information has been appended.

In addition, FSM_LOCAL_CHECK makes a copy of the distribution control information for each queue identifier found, and consolidates the user names for each queue identifier. That is, only one copy of the distribution is made for each queue, and the destination list contains all the destinations associated with that queue and service parm. FSM_LOCAL_CHECK also makes a (single) copy of the server object using the destination server name specified in the distribution control information.

If there is local-delivery queue information, FSM_LOCAL_CHECK signals DIRECTING_LOCALS back to the directing manager machine; otherwise, it signals DIRECTING_NO_LOCALS. If an exception was encountered while a copy of the server object was being made, this machine signals DIRECTING_LOCALS_COPY_EXPT.

FSM_NEXT_LOCAL_QUEUE

This FSM passes the next local-delivery queue identifier and a copy of the distribution to FSM_LOCAL_SCHED for processing. The input to this machine is a list of local delivery information that corresponds to the local users in the destination list, and copies of the distribution, one for each queue identifier. The local delivery information is obtained by the FSM_DIR_LOOKUP. The copies of the distribution are made by FSM_LOCAL_CHECK. When the list is exhausted, the signal END_QUEUE_IDS is sent to FSM_LOCAL_SCHED.

FSM_COUNT_EXCEPTS

This FSM scans the list of exception destinations created by FSM_LOCAL_SCHED or FSM_REMOTE_SCHED. If the list is empty, it reports SCHED_NO_EXCEPTS to FSM_LOCAL_SCHED or FSM_REMOTE_SCHED. Otherwise, it returns the signal SCHED_EXCEPTS. It is the mechanism used by the scheduling machines to decide whether to signal an exception to the directing manager or the routing manager.

FSM_DSU_CHECK

This FSM scans the DSU names in the destination list, and checks them against a table containing all the names of the local DSU. The input to this FSM is the destination list of the distribution that is being routed. If a match is found, this machine signals the routing manager machine with AT_LEAST_ONE_LOCAL; otherwise, it signals ALL_REMOTE.

FSM_RTG_LOOKUP

This machine scans the DSU names in the destination list and matches them against entries in the routing tables containing the next-DSU queue name and connection information. The input that it uses is the destination list of the distribution being routed. For each unique next-DSU queue name found, this machine associates with it all the DSUs mapped by the routing tables to that next-DSU queue. This machine decrements the hop count of the distribution copy so that loops in the network can be detected. It also makes a copy of the distribution for each unique next-DSU queue name found. The output of FSM_RTG_LOOKUP is the next-DSU queue information and the copies of the distribution.

This machine sends three signals back to the routing manager machine. RTG_LOOKUP_COMPLETE signifies that all DSUs in the destination list have been assigned a next-DSU queue. RTG_LOOKUP_COMPLETE_EXPTS means that at least one DSU in the destination list could not be assigned a queue, but at least one DSU in the list could be assigned a queue, and that it is recorded in a list of exception destinations.

RTG_LOOKUP_COMPLETE_ALL_EXPTS means that no DSU in the destination list could be routed.

FSM_NEXT_DSU

This machine scans the list of next-DSU queue names created by FSM_RTG_LOOKUP and returns the next queue name and a copy of the distribution to FSM_REMOTE_SCHED. The input to this machine is a list of queue names corresponding to the remote DSUs in the destination list, and copies of the distribution, one for each queue identifier. If no more names are in the list, it reports END_NEXT_DSUS back to FSM_REMOTE_SCHED.

Distribution Transport Sublayer—Format Set 2

DS_Send FSMs

Data Structures

DS_Send uses the following data structures:

MU_ID Registry: The MU_ID registry is a safe-stored data structure that records *MU_ID*-to-next-DSU-queue-entry mappings. It may be viewed as an array, with each element containing four entries. The first entry is an *MU_ID*. The second is a "distribution locator," which is used to locate the distribution associated with the entry's *MU_ID* on any next-DSU queue during error recovery processing. The third is the *MU_ID state*, and the fourth is the current *instance number*.

*MU_ID*s are assigned sequentially. Only contiguous blocks of entries in PURGED state containing the Registry's lowest *MU_ID*s and null distribution locators may be removed from the Registry automatically (and their space reclaimed). All other entries must be retained, unless explicitly removed by an operator.

Queue Entries: Each next-DSU queue entry maintains its *MU_ID* or a null value indicating that no *MU_ID* has been assigned.

Program Structure

A logical decomposition of DS_Send is given in Figure 45 on page 153. The following classes of routines exist in DS_Send:

- DS_Send Manager

  The only FSM in this class of routines is "DS_SEND_MANAGER" on page 154. It manages the transition between the LU 6.2 send and receive states, and also checks for the conversation idle condition, which causes the conversation to be deallocated.

- Send-State Manager

  The only FSM in this class of routines is "DS_SEND_SENDING" on page 156. This FSM is active whenever DS_Send is in the LU 6.2 send state, and it controls the sending of DMUs and CMUs to the partner DSU. The factors influencing this FSM include not only the MUs queued for the partner, but the partner DSU's flow control wishes (as indicated in the last

CRMU). These concepts are discussed in detail in "Throughput Control" on page 79.

- Receive-State Manager

  The only FSM in this class of routines is "DS_SEND_RECEIVING" on page 190. This routine manages DS_Send while in the LU 6.2 receive state. It repeatedly receives REMUs and CRMUs from the partner, parses them and signals the appropriate handler. The *terminate_conversation* bit from the CRMU is passed to the appropriate conversation control routine (even if the CRMU contains no *MU_ID*). Exception conditions (including receiving the send indication from the partner) are returned to the caller.

- Distribution Sending

  The only FSM in this class of routines is "DS_SEND_SEND_DIST" on page 159, which also implements part of the high-integrity actions. The distribution sending function requires only getting the distribution from the next-DSU queue and then examining it to determine if high-integrity actions or basic-integrity actions are required. If basic-integrity actions are required, this routine signals DS_SEND_SEND_DMU_NO_MU_ID. If high-integrity actions are required, this routine also assigns the *MU_ID* (if appropriate), and changes the *MU_ID state* to IN_TRANSIT before signalling DS_SEND_BUILD_SEND_DMU.

- Control MU Sending

  The only FSM in this class of routines is "DS_SEND_SEND_CONTROL_MU" on page 188. This routine is called repeatedly to read the next CMU from the control MU queue, send it, and discard it. If an exception occurs, the MU is not discarded, but is left on the queue to be resent later.

- REMU Receiving

  The only FSM in this class of routines is "DS_SEND_REMU_HANDLER" on page 204. This routine handles REMUs. The actions taken for a particular REMU depend on the status of the distribution (not found, retriable with a new *MU_ID*, or restartable with a DCMU), the *MU_ID state* (TRANSFER_PENDING, CQMU_PENDING, etc.), and the exception classification (not retriable or retriable). Potential actions include

  - Querying the partner to determine the Mid-MU Restart position.
  - Purging the *MU_ID* and retrying the distribution with a new *MU_ID* at a later time.
  - Terminating the distribution.

  REMUs with old *instance numbers* are simply discarded, as are REMUs reporting already-detected conversation failures and REMUs with no *MU_ID*.

- CRMU Receiving

  The only FSM in this class of routines is "DS_SEND_CRMU_HANDLER" on page 192. This routine processes the values from the CRMU's *MU_ID state* indication. The specific actions taken for a given distribution and CRMU depend on DS_Send's *MU_ID state*, the partner's *MU_ID state* as reported in the CRMU, and DS_Send's ability to re-attempt the transmission (with or without mid-MU restart). The actions include

  - Purging the distribution because the partner has accepted responsibility.
  - Marking the distribution to be restarted later using a DCMU.
  - Purging the *MU_ID* and retrying the distribution with a new *MU_ID* at a later time.
  - Querying the partner for an updated *MU_ID state*.
  - Reissuing a lost SEMU.
  - Issuing a SEMU to recover from a lost DMU.
  - Reissuing a lost PRMU.
  - Terminating the distribution.

  CRMUs with old *instance numbers* are simply discarded.

- Basic-Integrity Actions

  The only FSM in this class of routines is "DS_SEND_SEND_DMU_NO_MU_ID" on page 181. It builds and sends to the partner the basic-integrity distribution MU. When the *suffix* has been sent, the distribution is discarded. Exceptions are handled by "DS_SEND_EXCEPT_NO_MU_ID" on page 185.

- High-Integrity Actions

  These routines associate an *MU_ID* to a distribution, set the *MU_ID state* to IN_TRANSIT, encode the distribution into the appropriate DMU (DTMU, DCMU or DRMU), send it to the partner, and (in the absence of exception conditions) set the *MU_ID state* to TRANSFER_PENDING.

  The FSMs in this class are:

  - "DS_SEND_SEND_DIST" on page 159.

    This routine also implements the distribution sending function discussed above. The high-integrity actions it performs are to assign the distribution's *MU_ID*, if appropriate, and set the *MU_ID state* to IN_TRANSIT. It then signals DS_SEND_BUILD_SEND_DMU, which processes the distribution and returns the results of the transmission.

  - "DS_SEND_BUILD_SEND_DMU" on page 163.

    This routine encodes the distribution, reads the server object and sends the DMU to the partner DSU. Immediately before sending the *suffix*, it also changes the *MU_ID state* to TRANSFER_PENDING. Exceptions encountered during processing are passed to the appropriate exception-handling FSM. After the *suffix* has been sent, DS_SEND_CLEANUP_DMU is signalled to perform the required post-transmission processing.

  - "DS_SEND_CLEANUP_DMU" on page 166.

    This FSM simply replaces the distribution on the next-DSU queue. Exceptions cause the appropriate exception-handling FSM to be signalled.

- Pre-Sending Exception Actions

  The only FSM in this class of routines is "DS_SEND_DMU_EXCEPT_NOT_SENDING" on page 168. Exceptions occurring before the DMU's *prefix* is passed to LU 6.2 are known as pre-sending exceptions. There are three types of pre-sending exceptions:

  - A queue exception occurs on the READQ that gets the distribution from the next-DSU queue.
  - An MU_ID registry exception occurs on the inspection of the *MU_ID state*.
  - An MU_ID registry exception occurs when a new *MU_ID* is assigned to the distribution or an existing *MU_ID state* is set to IN_TRANSIT.

  All three exceptions result in the next-DSU queues being held, which prevents any other distribution transmissions from being attempted. When queue exceptions are encountered, CMU traffic continues, if possible; MU_ID registry exceptions cause the immediate termination of DS_Send.

- REMU Actions

  The actions that may be performed in response to a REMU are listed above, under "REMU Receiving." The FSMs in this class are:

  - "DS_SEND_QUERY_ON_REMU" on page 208.

    This FSM is signalled when a REMU reporting a retriable exception is received, mid-MU restart is possible, and the sender's *MU_ID state* is either TRANSFER_PENDING or RETRY_PENDING. In the former case (i.e., the sender's *MU_ID state* is TRANSFER_PENDING), DS_Send completed sending the distribution before receiving the partner's Prog_Error indication. In the latter case, DS_Send received the Prog_Error indication while sending the DMU, and set the *MU_ID state* to RETRY_PENDING in anticipation of receiving a REMU that reported a retriable exception.

    This FSM retains the distribution (i.e., it signals DS_SEND_RETAIN_DIST), holds the next-DSU queues, sets the *MU_ID state* to CQMU_PENDING, and issues a CQMU to query the partner.

  - "DS_SEND_RETRY_ON_REMU" on page 210.

    This FSM is signalled when the distribution in question is to be retried with a new *MU_ID*. To state these conditions more precisely, DS_Send's *MU_ID state* is TRANSFER_PENDING, CQMU_PENDING or RETRY_PENDING, mid-MU restart via a DCMU is not available, and the REMU from partner reports a retriable exception.

    Preparing a distribution for retry with a new *MU_ID* involves purging the (existing) *MU_ID* from the registry, sending a PRMU, and marking the distribution as having no *MU_ID*. The next-DSU queues are also held.

  - "DS_SEND_CHECK_CONV_FAIL" on page 212.

    This FSM determines if a just-received REMU is reporting an already-detected conversation failure. Conversation failures may cause the loss of a partial or complete MU, and are unusual in that the affected *MU_ID*s might be detected by either, neither, or both DS_Send and DS_Receive. Because either DS_Send or DS_Receive may be the only detector of the failure, they both initiate exception recovery actions by sending a SEMU or REMU (respectively). This leads to the conversation failures that are detected by both DS_Send and DS_Receive being reported by both a SEMU and a REMU. In such cases, the SEMU is preferred and the REMU is discarded.

(If an MU_ID affected by a conversation failure is detected by neither DS_Send nor DS_Receive, then either or both of the DSUs will eventually grow impatient at the *MU_ID state* being active. If DS_Receive grows impatient, it will send an unsolicited CRMU; if DS_Send grows impatient, it will send a CQMU.)

This FSM is called whenever the just-received REMU might be redundant, and returns either an indication that the REMU is redundant (and has thus been discarded) or the recovery action (Not Retriable, Retriable With Mid-MU, Retriable Without Mid-MU, or Retriable Retry Exhausted).

- Shared REMU/CRMU Actions

  The only FSM in this class of routines is "DS_SEND_TERMINATE_DIST" on page 214. It is called when a REMU or CRMU is received and DS_Send determines that an exception condition makes reattempting transmission of the distribution inappropriate. Terminating a distribution involves the following actions:

  – Holding the next-DSU queues (if requested to do so),
  – Generating a distribution report (if appropriate),
  – Setting the *MU_ID state* to PURGED,
  – Signalling DS_SEND_DISCARD_DIST.
  – Generating a PRMU to the partner.

- CRMU Actions

  The actions that may be performed in response to a CRMU are listed above, under "CRMU Receiving." The FSMs in this class are:

  – "DS_SEND_RELEASE_ON_CRMU" on page 196.

    This FSM is signalled under a variety of conditions and with a variety of input conditions, but its primary function is simply to release the distribution for later processing. A CQMU is generated, if requested.

  – "DS_SEND_PURGE_ON_CRMU" on page 198.

    This FSM is signalled when the partner DSU accepts responsibility for the distribution (i.e., a CRMU(COMPLETED) is received), or when a PRMU has been lost and a new PRMU must be generated. It changes the *MU_ID state* to PURGED and issues a PRMU to the partner. If requested, the distribution is also discarded.

  – "DS_SEND_RETRY_ON_CRMU" on page 200.

    If a distribution's transfer to the partner is interrupted by an exception, DS_Send determines whether to terminate the distribution or to attempt to recover from the exception. If DS_Send decides to attempt recovery, this FSM is signalled. It either suspends the distribution by setting the *MU_ID state* to SUSPENDED, or it purges the *MU_ID* and prepares the distribution to be retried later with a new *MU_ID*.

  - "DS_SEND_ISSUE_SEMU_ON_CRMU" on page 202.

    This FSM issues a SEMU and releases the distribution for further processing. It is called when DS_Send perceives a lost SEMU, or DMU. Lost SEMUs are merely regenerated.

- Basic-Integrity Exception Actions

  The only FSM in this class of routines is "DS_SEND_EXCEPT_NO_MU_ID" on page 185. It is signalled to process all exception conditions encountered during the transmission of basic-integrity distributions. Depending on the exception, the distribution is either discarded or retained to be retried (without mid-MU restart) later. The next-DSU queues are held, if appropriate.

- High-Integrity Exception Actions

  These FSMs handle the exception conditions detected in "DS_SEND_BUILD_SEND_DMU" on page 163. The FSMs in this class are:

  - "DS_SEND_DMU_ENCODE_EXCEPT" on page 170.

    This FSM is signalled by DS_SEND_BUILD_SEND_DMU and handles builder or server exceptions encountered *after* the first Send_Data LU 6.2 verb has been issued. A Send_Error LU 6.2 verb is issued, and DS_SEND_CLEANUP_EXCEPT is signalled to process the distribution and MU_ID.

  - "DS_SEND_CLEANUP_EXCEPT" on page 172.

    This FSM is signalled by DS_SEND_BUILD_SEND_DMU when an exception is encountered while building the first part of a DMU, or when a conversation failure is detected. It is also signalled by DS_SEND_DMU_ENCODE_EXCEPT. This FSM

    - Retains the distribution with or without holding the next-DSU queues, as appropriate.
    - Sets the *MU_ID state* to TERMINATION_PENDING or RETRY_PENDING, as appropriate.
    - Issues a SEMU to the partner describing the exception.

  - "DS_SEND_PROG_ERROR_RECEIVED" on page 174.

    This FSM is signalled whenever a Prog_Error indication is received from LU 6.2. It retains the distribution (without holding the next-DSU queues), and sets the *MU_ID state* to RETRY_PENDING.

  - "DS_SEND_DMU_PROTOCOL_ERROR" on page 176.

    This FSM is signalled if, while transmitting a DMU, DS_Send detects the partner DSU violating the DS usage of the LU 6.2 basic conversation protocol boundary. In this model, the affected distribution is retained, the *MU_ID state* set to TERMINATE_PENDING and the conversation immediately deallocated. Instead of merely deallocating the conversation, implementations may issue a Send_Error LU 6.2 verb, generate a SEMU, flush their control MU queue, and deallocate the conversation.

- "DS_SEND_MU_ID_STATE_ERROR" on page 178.

  This FSM is signalled if the *MU_ID state* cannot be set to TRANSFER_PENDING before the *suffix* is sent to the partner. The distribution is retained and the next-DSU queues are held.

- DS_Send Utility Routines

  These utilities are signalled by many of the DS_Send FSMs. The FSMs in this class are:

  - "DS_SEND_RETAIN_DIST" on page 216.

    When processing on a distribution is halted temporarily, this FSM is signalled to terminate any server operations in progress and to clear the distribution's in-use mark, making the distribution available for further processing. Retaining a distribution involves the following actions:

    - Holding the next-DSU queues, if requested.
    - Issuing a Terminate_Read verb to the server. The restartability of the server (if originally requested) is maintained.
    - Signalling the queue manager with RELEASEQ to remove the in-use mark from the distribution. The distribution will then be available to other DS_Send processes.

  - "DS_SEND_DISCARD_DIST" on page 218.

    Discarding a distribution involves the following actions:

    - Deleting the server object for this distribution, if one exists. This involves:
      - Issuing the Terminate_Read server verb if the server object was being read.
      - Issuing the Terminate_Restartability server verb if restartability had been requested.
      - Decrementing the DS lock count for the server object.
    - Signalling the queue manager with DEQ to remove the distribution from the next-DSU queue.

  - "DS_SEND_CONVERSATION_CONTROL" on page 222.

    This FSM records what DS_Send may send to the partner DSU at any particular time. For example, DS_Send

    - May be able to send only control MUs,
    - May be able to send both control MUs and a single distribution MU,
    - May be required to terminate the conversation after any waiting control MUs have been sent.

  - "DS_SEND_SEND_CONVERSATION_MGR" on page 220

    This FSM sends a single buffer to the partner DSU by issuing an LU 6.2 Send_Data verb. The outcome of the Send_Data (i.e., LU 6.2 accepted the data without detecting an exception, a conversation failure was detected, or the partner DSU issued a Send_Error) is returned to the caller.

- "DS_SEND_MU_ID_REGISTRY" on page 223

  This procedure manages DS_Send's access to the MU_ID registry. An *MU_ID state* may be assigned or inspected, or a just-received *instance number* may be compared to the *instance number* in the registry. (MUs containing an *instance number* lower than the registry's are ignored.)

- "UPM_CHECK_DUP_CONV_FAIL_REPORT" on page 224

  As discussed earlier (see the discussion for DS_SEND_CHECK_CONV_FAIL above, under "Program Structure" on page 146), some conversation failures cause both DS_Send and DS_Receive to initiate exception processing for the same *MU_ID*. This procedure is signalled by DS_SEND_CHECK_CONV_FAIL to determine if a given REMU does or does not represent such a duplicate (and extraneous) exception report for an already-known conversation failure.

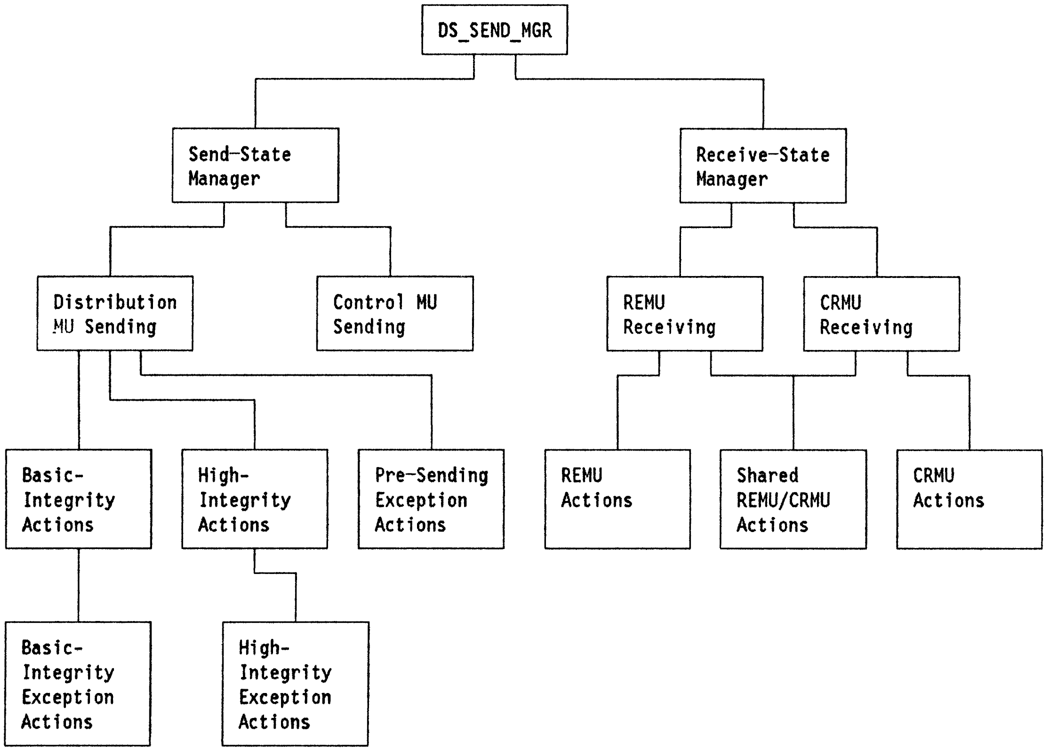


Figure 45. DS_Send Logical Structure

DS_SEND_MANAGER

Function: This finite-state machine describes the functional processing for DS_Send. It is started by the DS_Router_Director via a START_TRANSACTION signal, by LU 6.2 on receiving an Attach, or by the operator via START_TRANSACTION. If DS_Send is started by LU 6.2 via Attach, it is initially in the LU 6.2 receive state. Otherwise, it is initially in the LU 6.2 send state.

The primary processing of this FSM is done in states 4 (SEND), 5 (RCV) and 6 (IDLE). When in RCV state (state 5), DS_Send is in the LU 6.2 receive state and is receiving control MUs from the partner. If DS_Send then receives the change direction indication from LU 6.2, it immediately attempts to send any waiting MUs to the partner by signalling DS_SEND_SENDING.

When in SEND state (state 4), DS_Send is in the LU 6.2 send state, and is sending MUs to the partner. The lower-level FSMs signal DS_Send to enter the LU 6.2 receive state by returning CHANGE_DIRECTION. DS_SEND_MANAGER first checks to determine if an idle conversation exists by signalling IDLE_DETECTOR. If an idle conversation does exist, the conversation is simply deallocated. If the conversation is not idle, DS_Send goes into the LU 6.2 receive state by signalling DS_SEND_RECEIVING.

An idle conversation exists whenever both of the following conditions hold:

- When DS_Send was last in the LU 6.2 receive state, it received no CMUs from the partner.
- DS_Send, which is currently in the LU 6.2 send state, has no MUs (distribution or control MUs) to send to the partner DS_Receive.

Note: Upon detecting a conversation failure, implementations attempt to reactivate the session once.

This FSM gets control from one of the following:

- Signals from higher-level finite-state machines or procedures:
  - from "FSM_SCHED_MGR" on page 351
    - START_TRANSACTION
- Signals from lower-level DS_Send finite-state machines:
  - from "DS_SEND_SENDING" on page 156
    - CHANGE_DIRECTION
    - DEALLOC_LOCAL
    - DEALLOC_ABEND
    - DEALLOC_FLUSH
  - from "DS_SEND_RECEIVING" on page 190
    - CHANGE_DIRECTION
    - DEALLOC_LOCAL
    - DEALLOC_FLUSH
    - DEALLOC_ABEND
- Signals from machines providing common services:
  - from "IDLE_DETECTOR" on page 284
    - CHANGE_DIRECTION
    - DEALLOC_FLUSH
- Signals from LU 6.2 presentation services
  - ATTACH
  - OK
  - ALLOC_PARM_ERROR
  - ALLOCATION_ERROR

	States						
	RESET	**ALLOC**	**GET ATT**	**SEND**	**RCV**	**IDLE**	**END**
Inputs	**01**	**02**	**03**	**04**	**05**	**06**	**07**
START TRANSACTION	2a	/	/	/	/	/	/
ATTACH	3b	/	/	/	/	/	/
OK	/	4c	5f	/	/	/	1
ALLOC PARM ERROR	/	1d	/	/	/	/	/
ALLOCATION ERROR	/	7e	/	/	/	/	/
CHANGE DIRECTION	/	/	/	6g	4c	5f	/
DEALLOC LOCAL	/	/	/	7h	7h	/	/
DEALLOC FLUSH	/	/	/	7i	7i	7i	/
DEALLOC ABEND	/	/	/	7j	7j	/	/

Output Code	**Function**
a	Signal LU 6.2 presentation services with ALLOCATE to establish the conversation with DS_Receive.
b	Signal LU 6.2 presentation services with GET_ATTRIBUTES to determine the LU name and mode name of the partner.
c	Signal DS_SEND_SENDING with START.
d	Notify operations of the exception condition.
e	Notify operations of the exception condition. Signal LU 6.2 presentation services with DEALLOCATE specifying *type*(LOCAL).
f	Signal DS_SEND_RECEIVING with START.
g	Signal IDLE_DETECTOR with CHANGE_DIRECTION.
h	Signal LU 6.2 presentation services with DEALLOCATE specifying *type*(LOCAL).
i	Signal LU 6.2 presentation services with DEALLOCATE specifying *type*(FLUSH).
j	Signal LU 6.2 presentation services with DEALLOCATE specifying *type*(ABEND).

DS_SEND_SENDING

Function: This finite-state machine describes the functional processing for DS_Send while in the LU 6.2 send state. It first checks the flow control status of the connection, which will be ″send control MU,″ ″send distribution,″ or ″terminate conversation.″ If no control MUs are waiting and the flow control status is ″send distribution,″ then a distribution (DTMU, DRMU, or DCMU) is sent. The above sequence is repeated until the caller is signalled either to change direction, or deallocate the conversation.

Multiple CMUs may be sent both immediately before and after the distribution, before the change direction.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_MANAGER" on page 154
    - START
- Signals from lower-level DS_Send finite-state machines:
  - from "DS_SEND_CONVERSATION_CONTROL" on page 222
    - SEND_CONTROL_MU
    - SEND_DISTRIBUTION
    - TERMINATE_CONVERSATION
  - from "DS_SEND_SEND_CONTROL_MU" on page 188
    - CONTROL_MU_SENT
    - CONTROL_MU_QUEUE_EMPTY
    - PROTOCOL_ERROR
    - CONVERSATION_FAILURE
    - DS_SEND_SYSTEM_ERROR
    - PROG_ERROR
  - from "DS_SEND_SEND_DIST" on page 159
    - DISTRIBUTION_COMPLETE
    - DISTRIBUTION_QUEUES_EMPTY
    - PROTOCOL_ERROR
    - CONVERSATION_FAILURE
    - SEND_SIDE_EXCEPT
    - PROG_ERROR
    - DS_SEND_SYSTEM_ERROR

	States					
	RESET	CONV CNTL	CNTL MU FIRST	DIST	CNTL MU ONLY	CNTL MU FLUSH
Inputs	01	02	03	04	05	06
START	2a	/	/	/	/	/
SEND DISTRIBUTION	/	3b	/	/	/	/
SEND CONTROL MU	/	5b	/	/	/	/
TERMINATE CONVERSATION	/	6b	/	/	/	/
CONTROL MU SENT	/	/	2a	/	2a	2a
CONTROL MU QUEUE EMPTY	/	/	4d	/	1c	1g
DISTRIBUTION COMPLETE	/	/	/	2a	/	/
DISTRIBUTION QUEUES EMPTY	/	/	/	1c	/	/
PROTOCOL ERROR	/	/	1e	1e	1e	1e
CONVERSATION FAILURE	/	/	1f	1f	1f	1f
SEND SIDE EXCEPT	/	/	/	2a	/	/
PROG ERROR	/	/	1c	1c	1c	1c
DS SEND SYSTEM ERROR	/	/	1e	1e	1e	1e

Output Code	Function
a	Signal DS_SEND_CONVERSATION_CONTROL with QUERY_FLOW_CONTROL.
b	Signal DS_SEND_SEND_CONTROL_MU with SEND.
c	Signal caller with CHANGE_DIRECTION.
d	Signal DS_SEND_SEND_DIST with START.
e	Signal caller with DEALLOCATE_ABEND for protocol or system errors.
f	Signal caller with DEALLOCATE_LOCAL.
g	Signal caller with DEALLOCATE_FLUSH.

DS_SEND_SEND_DIST

Function: This finite-state machine initiates the transmission of a distribution to the partner. If the distribution requires only basic integrity, then DS_SEND_SEND_DMU_NO_MU_ID is signalled to transmit the distribution. If high integrity is required, then the *MU_ID state* is changed to IN_TRANSIT, and DS_SEND_BUILD_SEND_DMU is signalled.

A failure on the initial READQ will cause a HOLD to be placed on the next-DSU queues and a SEND_SIDE_EXCEPT to be returned to the caller, who will continue sending CMUs. Eventually, all CMUs will be transmitted and this routine will again be called to send a distribution. At this point, the HOLD placed on the next-DSU queues will cause this routine to return DISTRIBUTION_QUEUES_EMPTY.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from “DS_SEND_SENDING” on page 156
    - START
- Signals from lower-level DS_Send finite-state machines:
  - from “DS_SEND_DMU_EXCEPT_NOT_SENDING” on page 168
    - SEND_SIDE_EXCEPT
  - from “DS_SEND_SEND_DMU_NO_MU_ID” on page 181
    - PROG_ERROR
    - CONVERSATION_FAILURE
    - PROTOCOL_ERROR
    - SEND_SIDE_EXCEPT
    - DISTRIBUTION_COMPLETE
  - from “DS_SEND_BUILD_SEND_DMU” on page 163
    - PROG_ERROR
    - CONVERSATION_FAILURE
    - PROTOCOL_ERROR
    - SEND_SIDE_EXCEPT
    - DISTRIBUTION_COMPLETE
    - DS_SEND_SYSTEM_ERROR
  - from “DS_SEND_MU_ID_REGISTRY” on page 223
    - NOT_ASSIGNED
    - SUSPENDED
    - MU_ID_NOT_USED
    - MU_ID_OP_OK
    - MU_ID_OP_NOT_OK
- Signals from machines providing common services:
  - from “QUEUE_MGR” on page 357
    - QUEUE_OK
    - QUEUE_EMPTY
    - QUEUE_NOT_OK

Inputs	States				
	RESET	RDQ DIST	MU_ID STATE	ASSGN MU_ID	PROC DMU
	01	02	03	04	05
START	2a	/	/	/	/
QUEUE OK	/	3b	/	/	/
QUEUE EMPTY	/	1c	/	/	/
QUEUE NOT OK	/	5d	/	/	/
NOT ASSIGNED	/	/	4e	/	/
SUSPENDED	/	/	4f	/	/
MU ID NOT USED	/	/	5g	/	/
MU ID OP OK	/	/	/	5i	/
MU ID OP NOT OK	/	/	5h	5h	/
PROG ERROR	/	/	/	/	1j
CONVERSATION FAILURE	/	/	/	/	1k
PROTOCOL ERROR	/	/	/	/	1m
DISTRIBUTION COMPLETE	/	/	/	/	1n
SEND SIDE EXCEPT	/	/	/	/	1o
DS SEND SYSTEM ERROR	/	/	/	/	1p

Output Code	Function
a	Signal QUEUE_MGR with READQ specifying *queue*(NEXT-DSU-QUEUE) *where_MU_ID_state_is*(NOT_ASSIGNED, or SUSPENDED) to get the next distribution to be sent. The HOLD flag is respected for this operation; if the HOLD indication is set for this queue, QUEUE_MGR returns QUEUE_EMPTY, even if other queue exception conditions are present.
b	Signal DS_SEND_MU_ID_REGISTRY with INSPECT. If the distribution uses basic integrity, processing continues with the MU_ID_NOT_USED input.
c	Signal caller with DISTRIBUTION_QUEUES_EMPTY.
d	Signal DS_SEND_DMU_EXCEPT_NOT_SENDING with QUEUE_ACCESS_EXCEPT.
e	Signal DS_SEND_MU_ID_REGISTRY with ASSIGN.
f	Signal DS_SEND_MU_ID_REGISTRY with IN_TRANSIT to set the *MU_ID state* and update the *instance number*.
g	Signal DS_SEND_SEND_DMU_NO_MU_ID with START.
h	Signal DS_SEND_DMU_EXCEPT_NOT_SENDING with REGISTRY_EXCEPT.
i	Signal DS_SEND_BUILD_SEND_DMU with START.
j	Signal caller with PROG_ERROR.
k	Signal caller with CONVERSATION_FAILURE to indicate that an error has occurred in the conversation between DS_Send and DS_Receive.
m	Signal caller with PROTOCOL_ERROR.
n	Signal caller with DISTRIBUTION_COMPLETE to indicate that the MU has been encoded and sent to DS_Receive.
o	Signal caller with SEND_SIDE_EXCEPT.
p	Signal caller with DS_SEND_SYSTEM_ERROR.

DS_SEND_BUILD_SEND_DMU

Function: This finite-state machine encodes and sends a high-integrity distribution. States 3 and 4 are a loop which builds and sends the distribution pieces. States 4, 5 and 6 are a loop which reads, builds and sends the server object pieces Before sending the *suffix*, the *MU_ID state* is changed to TRANSFER_PENDING.

This FSM gets control from one of the following:

- Signals from higher-level finite-state machines or procedures:
  - from "DS_SEND_SEND_DIST" on page 159
    - START
  - from the operator
    - OP_SUSP
- Signals from lower-level DS_Send finite-state machines:
  - from "DS_SEND_SEND_CONVERSATION_MGR" on page 220
    - OK
    - PROG_ERROR
    - PROTOCOL_ERROR
    - CONVERSATION_FAILURE
  - from "DS_SEND_CLEANUP_DMU" on page 166
    - DISTRIBUTION_COMPLETE
    - SEND_SIDE_EXCEPT
  - from "DS_SEND_DMU_ENCODE_EXCEPT" on page 170
    - PROG_ERROR
    - CONVERSATION_FAILURE
    - PROTOCOL_ERROR
    - SEND_SIDE_EXCEPT
    - DS_SEND_SYSTEM_ERROR
  - from "DS_SEND_PROG_ERROR_RECEIVED" on page 174
    - PROG_ERROR
  - from "DS_SEND_DMU_PROTOCOL_ERROR" on page 176
    - PROTOCOL_ERROR
  - from "DS_SEND_CLEANUP_EXCEPT" on page 172
    - DS_SEND_SYSTEM_ERROR
    - SEMU_SENT
  - from "DS_SEND_MU_ID_STATE_ERROR" on page 178
    - DS_SEND_SYSTEM_ERROR
    - SEMU_SENT
  - from "DS_SEND_MU_ID_REGISTRY" on page 223
    - MU_ID_OP_OK
    - MU_ID_OP_NOT_OK
- Signals from machines providing common services:
  - from "IDLE_DETECTOR" on page 284
    - no signals returned from this FSM.
  - from "SERVER_MGR" on page 354
    - OBJECT_OK
    - NO_OBJECT_EXISTS
    - OBJECT_EOD
    - OBJECT_NOT_OK
- from "BUILDER" on page 359
  - BUILD_OK
  - BUILD_OK_GET_OBJECT
  - BUILD_COMPLETE
  - BUILD_NOT_OK

	States									
	RESET	ENC FIRST	SEND BUILD LOOP	BUILD	SEND READ BUILD LOOP	GET OBJ	TRANS PEND	SEND LAST	CLEAN UP	CONV FAIL
Inputs	01	02	03	04	05	06	07	08	09	10
START	2a	/	/	/	/	/	/	/	/	/
BUILD OK	/	3b	/	3g	/	/	/	/	/	/
BUILD OK GET OBJECT	/	5b	/	5g	/	/	/	/	/	/
BUILD COMPLETE	/	7c	/	7h	/	/	/	/	/	/
BUILD NOT OK	/	9d	/	9i	/	/	/	/	/	/
OK	/	/	4a	/	6j	/	/	9m	/	/
PROG ERROR	/	/	9e	/	9e	/	/	9e	1n	/
CONVERSATION FAILURE	/	/	10d	/	10d	/	/	10d	1o	/
PROTOCOL ERROR	/	/	9f	/	9f	/	/	9f	1p	/
OBJECT OK	/	/	/	/	/	4a	/	/	/	/
NO OBJECT EXISTS	/	/	/	/	/	4k	/	/	/	/
OBJECT EOD	/	/	/	/	/	4k	/	/	/	/
OBJECT NOT OK	/	/	/	/	/	9i	/	/	/	/
MU ID OP OK	/	/	/	/	/	/	8g	/	/	/
MU ID OP NOT OK	/	/	/	/	/	/	9t	/	/	/
DISTRIBUTION COMPLETE	/	/	/	/	/	/	/	/	1q	/
SEND SIDE EXCEPT	/	/	/	/	/	/	/	/	1r	/
DS SEND SYSTEM ERROR	/	/	/	/	/	/	/	/	1s	1o
SEMU SENT	/	/	/	/	/	/	/	/	1r	1o
OP SUSP	-	9d	9i	9i	9i	9i	-	-	-	-

Output Code	Function
a	Signal BUILDER with BUILD to start or continue building the DTMU, DRMU or DCMU.
b	Signal IDLE_DETECTOR with SOMETHING_SENT. Signal DS_SEND_SEND_CONVERSATION_MGR with SEND_BUFFER to send the MU information to LU 6.2.
c	Signal IDLE_DETECTOR with SOMETHING_SENT. Signal DS_SEND_MU_ID_REGISTRY with TRANSFER_PEND.
d	Signal DS_SEND_CLEANUP_EXCEPT with START to clean up the distribution and generate a SEMU. (But no Send_Error will be issued.)
e	Signal DS_SEND_PROG_ERROR_RECEIVED with PROG_ERROR_RECEIVED to clean up the distribution before handling the PROG_ERROR from the partner DSU.
f	Signal DS_SEND_DMU_PROTOCOL_ERROR with PROT_ERROR_DETECTED to clean up the distribution before handling the protocol error from the partner DSU.
g	Signal DS_SEND_SEND_CONVERSATION_MGR with SEND_BUFFER to send the MU information to LU 6.2.
h	Signal DS_SEND_MU_ID_REGISTRY with TRANSFER_PEND.
i	Signal DS_SEND_DMU_ENCODE_EXCEPT with START to take appropriate error-handling actions.
j	Signal SERVER_MGR with READ to read the object and perform any initialization for reading, if not yet performed.
k	Signal BUILDER with END_OBJECT to indicate that the server has returned EOD and the last segment of the object should be built, or, if no object exists, the length of the data will be 0.
m	Signal DS_SEND_CLEANUP_DMU with START.
n	Signal caller with PROG_ERROR.
o	Signal caller with CONVERSATION_FAILURE to indicate that an error has occurred in the conversation between DS_Send and DS_Receive.
p	Signal caller with PROTOCOL_ERROR.
q	Signal caller with DISTRIBUTION_COMPLETE to indicate that the MU has been encoded and sent to DS_Receive.
r	Signal caller with SEND_SIDE_EXCEPT.
s	Signal caller with DS_SEND_SYSTEM_ERROR.
t	Signal DS_SEND_MU_ID_STATE_ERROR with START to clean up the distribution after the failed attempt to set the *MU_ID state* to TRANSFER_PENDING.

DS_SEND_CLEANUP_DMU

Function: This finite-state machine describes the functional processing for tidying up a distribution once it has been sent to the partner and the *MU_ID state* set to TRANSFER_PENDING.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_BUILD_SEND_DMU" on page 163
    - START
- Signals from lower-level DS_Send finite-state machines:
  - from "DS_SEND_RETAIN_DIST" on page 216
    - DIST_RETAINED
    - RELQ_FAILED
    - TERM_READ_FAILED
    - TERM_READ_RELQ_FAILED
  - from "DS_SEND_CONVERSATION_CONTROL" on page 222
    - No signals are returned from this FSM.
- Signals from machines providing common services:
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK

	States		
	RESET	RETAIN DIST	HOLDQ
Inputs	01	02	03
START	2a	/	/
DIST RETAINED	/	1b	/
RELQ FAILED	/	3c	/
TERM READ FAILED	/	3c	/
TERM READ RELQ FAILED	/	3c	/
QUEUE OK	/	/	1d
QUEUE NOT OK	/	/	1d

Output Code	Function
a	Signal DS_SEND_CONVERSATION_CONTROL with DMU_SENT. Signal DS_SEND_RETAIN_DIST with NO_HOLDQ.
b	Signal caller with DISTRIBUTION_COMPLETE to indicate that the MU has been encoded and sent to DS_Receive.
c	Signal QUEUE_MGR with HOLD to place an exception-hold on all next-DSU queues for this connection; no more distributions are sent to the partner DSU. The control MU queue is not held.
d	Signal caller with SEND_SIDE_EXCEPT.

DS_SEND_DMU_EXCEPT_NOT_SENDING

Function: This finite-state machine describes the functional processing for non-builder exceptions detected in a DMU before any portion of the DMU is transferred to the partner DSU. Pre-sending exceptions are of three types:

- A queue exception occurs on the READQ that gets the distribution from the next-DSU queue.
- An MU_ID registry exception occurs on the inspection of the *MU_ID state*.
- An MU_ID registry exception occurs when a new *MU_ID* is assigned to the distribution or an existing *MU_ID state* is set to IN_TRANSIT.

All three exceptions result in an exception-hold being placed on the next-DSU queues, which prevents any distributions from being sent. If this FSM is signalled because of a queue exception, and if the next-DSU queue hold operation completes successfully, then SEND_SIDE_EXCEPT is returned, and CMUs will continue to be exchanged with the partner. If the hold operation on the next-DSU queues fails, or this FSM is signalled because of an MU_ID registry exception, DS_SEND_SYSTEM_ERROR is returned, and the conversation is terminated immediately.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_SEND_DIST" on page 159
    - REGISTRY_EXCEPT
    - QUEUE_ACCESS_EXCEPT
- Signals from machines providing common services:
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK

	States			
	RESET	REG ERR HOLD	RELQ	QUE ERR HOLD
Inputs	01	02	03	04
REGISTRY EXCEPT	2a	/	/	/
QUEUE ACCESS EXCEPT	4a	/	/	/
QUEUE OK	/	3b	1c	1d
QUEUE NOT OK	/	3b	1c	1c

Output Code	Function
a	Signal QUEUE_MGR with HOLD to place an exception-hold on all next-DSU queues for this connection; no more distributions are sent to the partner DSU. The control MU queue is not held.
b	Signal QUEUE_MGR with RELEASEQ to remove the in-use mark from the entry on the next-DSU queue. Following the RELEASEQ, the entry will be available for processing.
c	Notify operations of the exception condition. Signal caller with DS_SEND_SYSTEM_ERROR.
d	Notify operations of the exception condition. Signal caller with SEND_SIDE_EXCEPT.

DS_SEND_DMU_ENCODE_EXCEPT

Function: This finite-state machine describes the functional processing for errors detected while the DMU is being transferred to the partner DSU.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_BUILD_SEND_DMU" on page 163
    - START
- Signals from lower-level DS_Send finite-state machines:
  - from "DS_SEND_CLEANUP_EXCEPT" on page 172.
    - SEMU_SENT
    - DS_SEND_SYSTEM_ERROR
- Signals from LU 6.2 presentation services
  - OK
  - ALLOCATION_ERROR
  - DEALLOCATE_ABEND
  - PROG_ERROR
  - SVC_ERROR
  - RESOURCE_FAILURE

	States					
	RESET	SEND ERR	SEND SEMU	SEMU CONV FAIL	SEMU PROG ERR	SEMU PROT ERR
Inputs	01	02	03	04	05	06
START	2a	/	/	/	/	/
OK	/	3b	/	/	/	/
RESOURCE FAILURE	/	4b	/	/	/	/
PROG ERROR	/	5b	/	/	/	/
SVC ERROR	/	6b	/	/	/	/
ALLOC ERROR	/	4b	/	/	/	/
DEALLOC ABEND	/	4b	/	/	/	/
SEMU SENT	/	/	1c	1e	1f	1g
DS SEND SYSTEM ERROR	/	/	1d	1e	1d	1d

Output Code	Function
a	Signal LU 6.2 presentation services with SEND_ERROR.
b	Signal DS_SEND_CLEANUP_EXCEPT with START.
c	Notify operations of the exception condition. Signal caller with SEND_SIDE_EXCEPT.
d	Notify operations of the exception condition. Signal caller with DS_SEND_SYSTEM_ERROR.
e	Notify operations of the exception condition. Signal caller with CONVERSATION_FAILURE.
f	Notify operations of the exception condition. Signal caller with PROG_ERROR.
g	Notify operations of the exception condition. Signal caller with PROTOCOL_ERROR.

DS_SEND_CLEANUP_EXCEPT

Function: This finite-state machine describes the functional processing for cleaning up a distribution after an exception has been detected during the MU's transfer to the partner DSU. This FSM

- Retains the distribution with or without holding the next-DSU queues, as appropriate.
- Sets the *MU_ID state* to TERMINATION_PENDING or RETRY_PENDING, as appropriate.
- Issues a SEMU to the partner describing the exception.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_DMU_ENCODE_EXCEPT" on page 170
    - START
  - from "DS_SEND_BUILD_SEND_DMU" on page 163
    - START
- Signals from lower-level DS_Send finite-state machines:
  - from "DS_SEND_MU_ID_REGISTRY" on page 223
    - MU_ID_OP_OK
    - MU_ID_OP_NOT_OK
  - from "DS_SEND_RETAIN_DIST" on page 216
    - DIST_RETAINED
    - RELQ_FAILED
    - TERM_READ_FAILED
    - TERM_READ_RELQ_FAILED
    - HOLDQ_FAILED
    - HOLDQ_RELQ_FAILED
    - HOLDQ_TERM_READ_FAILED
    - HOLDQ_TERM_READ_RELQ_FAILED
- Signals from machines providing common services:
  - from "UPM_EXCEPT_RECOVERY_ACTION" on page 285
    - NOT_RETRIABLE
    - RETRIABLE_WITHOUT_MID_MU
    - RETRIABLE_WITH_MID_MU
    - RETRIABLE_RETRY_EXHAUSTED
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK

	States						
	RESET	RETRY ACT	RETAIN DIST NO RETRY	RETAIN DIST RETRY	MU_ID STATE	SEMU	SEMU SYS ERR
Inputs	01	02	03	04	05	06	07
START	2a	/	/	/	/	/	/
NOT RETRIABLE	/	3b	/	/	/	/	/
RETRIABLE WITHOUT MID MU	/	4e	/	/	/	/	/
RETRIABLE WITH MID MU	/	4e	/	/	/	/	/
RETRIABLE RETRY EXHAUSTED	/	3e	/	/	/	/	/
DIST RETAINED	/	/	5d	5c	/	/	/
RELQ FAILED	/	/	5d	5c	/	/	/
TERM READ FAILED	/	/	5d	5c	/	/	/
TERM READ RELQ FAILED	/	/	5d	5c	/	/	/
HOLDQ FAILED	/	/	5d	5c	/	/	/
HOLDQ RELQ FAILED	/	/	5d	5c	/	/	/
HOLDQ TERM READ FAILED	/	/	5d	5c	/	/	/
HOLDQ TERM READ RELQ FAILED	/	/	5d	5c	/	/	/
MU ID OP OK	/	/	/	/	6f	/	/
MU ID OP NOT OK	/	/	/	/	7f	/	/
QUEUE OK	/	/	/	/	/	1g	1h
QUEUE NOT OK	/	/	/	/	/	1h	1h

Output Code	Function
a	Signal UPM_EXCEPT_RECOVERY_ACTION with the exception code.
b	Signal DS_SEND_RETAIN_DIST with NO_HOLDQ.
c	Signal DS_SEND_MU_ID_REGISTRY with RETRY_PEND.
d	Signal DS_SEND_MU_ID_REGISTRY with TERM_PEND.
e	Signal DS_SEND_RETAIN_DIST with HOLDQ.
f	Build a SEMU describing the exception and signal QUEUE_MGR with WRITEQ specifying *queue*(CONTROL_MU_QUEUE). Implementations may choose to suppress this SEMU if the exception was an LU 6.2 ALLOCATION_ERROR. In this case, processing continues as though QUEUE_MGR returned QUEUE_OK.
g	Signal caller with SEMU_SENT.
h	Notify operations of the exception condition. Signal caller with DS_SEND_SYSTEM_ERROR.

DS_SEND_PROG_ERROR_RECEIVED

Function: This finite-state machine describes the actions taken to suspend the transmission of a distribution upon the receipt of a Prog_Error indication from the partner DSU. The distribution is retained (without holding the next-DSU queues), and the *MU_ID state* is set to RETRY_PENDING.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_BUILD_SEND_DMU" on page 163
    - PROG_ERROR_RECEIVED
- Signals from lower-level DS_Send finite-state machines:
  - from "DS_SEND_RETAIN_DIST" on page 216
    - DIST_RETAINED
    - RELQ_FAILED
    - TERM_READ_FAILED
    - TERM_READ_RELQ_FAILED
    - HOLDQ_FAILED
    - HOLDQ_RELQ_FAILED
    - HOLDQ_TERM_READ_FAILED
    - HOLDQ_TERM_READ_RELQ_FAILED
  - from "DS_SEND_MU_ID_REGISTRY" on page 223
    - MU_ID_OP_OK
    - MU_ID_OP_NOT_OK

	States		
	RESET	RETAIN DIST	RETRY PEND
Inputs	01	02	03
PROG ERROR RECEIVED	2a	/	/
DIST RETAINED	/	3b	/
RELQ FAILED	/	3b	/
TERM READ FAILED	/	3b	/
TERM READ RELQ FAILED	/	3b	/
HOLDQ FAILED	/	3b	/
HOLDQ RELQ FAILED	/	3b	/
HOLDQ TERM READ FAILED	/	3b	/
HOLDQ TERM READ RELQ FAILED	/	3b	/
MU ID OP OK	/	/	1c
MU ID OP NOT OK	/	/	1d

Output Code	Function
a	Signal DS_SEND_RETAIN_DIST with NO_HOLDQ.
b	Signal DS_SEND_MU_ID_REGISTRY with RETRY_PEND.
c	Notify operations of the exception condition. Signal caller with PROG_ERROR.
d	Notify operations of the exception condition. Signal caller with DS_SEND_SYSTEM_ERROR.

DS_SEND_DMU_PROTOCOL_ERROR

Function: This finite-state machine describes the actions taken if a violation of the DS usage of the LU 6.2 basic conversation verbs is detected from the partner DSU. The next-DSU queues are held, and the *MU_ID state* changed to TERMINATION_PENDING.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_BUILD_SEND_DMU" on page 163
    - PROT_ERROR_DETECTED
- Signals from lower-level DS_Send finite-state machines:
  - from "DS_SEND_RETAIN_DIST" on page 216
    - DIST_RETAINED
    - RELQ_FAILED
    - TERM_READ_FAILED
    - TERM_READ_RELQ_FAILED
    - HOLDQ_FAILED
    - HOLDQ_RELQ_FAILED
    - HOLDQ_TERM_READ_FAILED
    - HOLDQ_TERM_READ_RELQ_FAILED
  - from "DS_SEND_MU_ID_REGISTRY" on page 223
    - MU_ID_OP_OK
    - MU_ID_OP_NOT_OK

	States		
	RESET	RETAIN DIST	TERM PEND
Inputs	01	02	03
PROT ERROR DETECTED	2a	/	/
DIST RETAINED	/	3b	/
RELQ FAILED	/	3b	/
TERM READ FAILED	/	3b	/
TERM READ RELQ FAILED	/	3b	/
HOLDQ FAILED	/	3b	/
HOLDQ RELQ FAILED	/	3b	/
HOLDQ TERM READ FAILED	/	3b	/
HOLDQ TERM READ RELQ FAILED	/	3b	/
MU ID OP OK	/	/	1c
MU ID OP NOT OK	/	/	1d

Output Code	Function
a	Signal DS_SEND_RETAIN_DIST with HOLDQ.
b	Signal DS_SEND_MU_ID_REGISTRY with TERM_PEND.
c	Notify operations of the exception condition. Signal caller with PROTOCOL_ERROR.
d	Notify operations of the exception condition. Signal caller with DS_SEND_SYSTEM_ERROR.

DS_SEND_MU_ID_STATE_ERROR

Function: This finite-state machine is signalled if an MU_ID Registry exception prevents an *MU_ID state* from being set to TRANSFER_PENDING before the *suffix* is sent to the partner. The distribution is retained, and an exception-hold is placed on the next-DSU queues. DS_SEND_SYSTEM_ERROR is returned, which will eventually cause the conversation to be deallocated.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_BUILD_SEND_DMU" on page 163
    - START
- Signals from lower-level DS_Send finite-state machines:
  - from "DS_SEND_RETAIN_DIST" on page 216
    - DIST_RETAINED
    - RELQ_FAILED
    - TERM_READ_FAILED
    - TERM_READ_RELQ_FAILED
    - HOLDQ_FAILED
    - HOLDQ_RELQ_FAILED
    - HOLDQ_TERM_READ_FAILED
    - HOLDQ_TERM_READ_RELQ_FAILED

	States	
	RESET	**RETAIN DIST**
Inputs	**01**	**02**
START	2a	/
DIST RETAINED	/	1b
RELQ FAILED	/	1b
TERM READ FAILED	/	1b
TERM READ RELQ FAILED	/	1b
HOLDQ FAILED	/	1b
HOLDQ RELQ FAILED	/	1b
HOLDQ TERM READ FAILED	/	1b
HOLDQ TERM READ RELQ FAILED	/	1b

Output Code	Function
a	Signal DS_SEND_RETAIN_DIST with HOLDQ, specifying that an exception-hold is to be placed on the next-DSU queues.
b	Notify operations of the exception condition. Signal caller with DS_SEND_SYSTEM_ERROR.

DS_SEND_SEND_DMU_NO_MU_ID

Function: This finite-state machine describes the functional processing for controlling the encoding and sending of a basic integrity DMU. States 3 and 4 represent a loop that repeatedly builds and sends parts of the distribution. States 4, 5 and 6 represent a loop that repeatedly reads from the server and builds and sends the server object. After the *suffix* has been sent successfully, the distribution is discarded (without confirmation of it being accepted by the partner). Exceptions are handled by "DS_SEND_EXCEPT_NO_MU_ID" on page 185.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_SEND_DIST" on page 159
    - START
- Signals from lower-level DS_Send finite-state machines:
  - from "DS_SEND_DISCARD_DIST" on page 218
    - DIST_DISCARDED
    - QUEUE_FAILED
    - SERVER_FAILED
    - QUEUE_AND_SERVER_FAILED
  - from "DS_SEND_EXCEPT_NO_MU_ID" on page 185
    - SEND_SIDE_EXCEPT
    - PROG_ERROR
    - PROTOCOL_ERROR
    - CONVERSATION_FAILURE
- Signals from machines providing common services:
  - from "DS_SEND_SEND_CONVERSATION_MGR" on page 220
    - OK
    - PROG_ERROR
    - PROTOCOL_ERROR
    - CONVERSATION_FAILURE
  - from "IDLE_DETECTOR" on page 284
    - no signals returned from this FSM.
  - from "SERVER_MGR" on page 354
    - OBJECT_OK
    - NO_OBJECT_EXISTS
    - OBJECT_EOD
    - OBJECT_NOT_OK
- from "BUILDER" on page 359:
  - BUILD_OK
  - BUILD_OK_GET_OBJECT
  - BUILD_COMPLETE
  - BUILD_COMPLETE_NO_DATA
  - BUILD_NOT_OK

	States								
	RESET	ENC FIRST	SEND, BUILD, LOOP	BUILD	SEND, READ, BUILD, LOOP	READ OBJ	SEND LAST	CLEAN UP	ERR HAND
Inputs	01	02	03	04	05	06	07	08	09
START	2a	/	/	/	/	/	/	/	/
BUILD OK	/	3b	/	3g	/	/	/	/	/
BUILD OK GET OBJECT	/	5b	/	5g	/	/	/	/	/
BUILD COMPLETE	/	7b	/	7g	/	/	/	/	/
BUILD NOT OK	/	9c	/	9i	/	/	/	/	/
OK	/	/	4a	/	6j	/	8h	/	/
PROG ERROR	/	/	9d	/	9d	/	9d	/	1m
CONVERSATION FAILURE	/	/	9e	/	9e	/	9e	/	1n
PROTOCOL ERROR	/	/	9f	/	9f	/	9f	/	1o
OBJECT OK	/	/	/	/	/	4a	/	/	/
NO OBJECT EXISTS	/	/	/	/	/	4k	/	/	/
OBJECT EOD	/	/	/	/	/	4k	/	/	/
OBJECT NOT OK	/	/	/	/	/	9i	/	/	/
SEND SIDE EXCEPT	/	/	/	/	/	/	/	/	1p
DIST DISCARDED	/	/	/	/	/	/	/	1q	/
QUEUE FAILED	/	/	/	/	/	/	/	1q	/
SERVER FAILED	/	/	/	/	/	/	/	1q	/
QUEUE AND SERVER FAILED	/	/	/	/	/	/	/	1q	/

Output Code	**Function**
a	Signal BUILDER with BUILD to start building the DTMU.
b	Signal IDLE_DETECTOR with SOMETHING_SENT. Signal DS_SEND_SEND_CONVERSATION_MGR with SEND_BUFFER to send the MU information to LU 6.2.
c	Signal DS_SEND_EXCEPT_NO_MU_ID with ERROR_NO_DATA_SENT.
d	Signal DS_SEND_EXCEPT_NO_MU_ID with PROG_ERR_RCVD.
e	Signal DS_SEND_EXCEPT_NO_MU_ID with CONV_FAIL_RCVD.
f	Signal DS_SEND_EXCEPT_NO_MU_ID with PROTO_ERR_RCVD.
g	Signal DS_SEND_SEND_CONVERSATION_MGR with SEND_BUFFER to send the MU information to LU 6.2.
h	Signal DS_SEND_CONVERSATION_CONTROL with DMU_SENT. Signal DS_SEND_DISCARD_DIST with START.
i	Signal DS_SEND_EXCEPT_NO_MU_ID with SEND_SIDE_EXCEPT.
j	Signal SERVER_MGR with READ to read the object and perform any initialization for reading, if not yet performed.
k	Signal BUILDER with END_OBJECT to indicate that the server has returned EOD and the last segment of the object should be built, or if no object exists the length of the data will be 0.
m	Signal caller with PROG_ERROR.
n	Signal caller with CONVERSATION_FAILURE to indicate that an error has occurred in the conversation between DS_Send and DS_Receive.
o	Signal caller with PROTOCOL_ERROR.
p	Signal caller with SEND_SIDE_EXCEPT.
q	Signal caller with DISTRIBUTION_COMPLETE to indicate that the MU has been encoded and sent to DS_Receive.

DS_SEND_EXCEPT_NO_MU_ID

Function: This finite-state machine describes the functional processing for errors detected while a basic-integrity DMU is being transferred to the partner DSU. Depending on the exception, the distribution is either discarded or retained to be retried (without mid-MU restart) later. The next-DSU queues are held, if appropriate.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_SEND_DMU_NO_MU_ID" on page 181
    - SEND_SIDE_EXCEPT
    - ERROR_NO_DATA_SENT
    - CONV_FAIL_RCVD
    - PROG_ERR_RCVD
    - PROTO_ERR_RCVD
- Signals from lower-level DS_Send finite-state machines:
  - from "DS_SEND_DISCARD_DIST" on page 218
    - DIST_DISCARDED
    - QUEUE_FAILED
    - SERVER_FAILED
    - QUEUE_AND_SERVER_FAILED
  - from "DS_SEND_RETAIN_DIST" on page 216
    - DIST_RETAINED
    - RELQ_FAILED
    - TERM_READ_FAILED
    - TERM_READ_RELQ_FAILED
    - HOLDQ_FAILED
    - HOLDQ_RELQ_FAILED
    - HOLDQ_TERM_READ_FAILED
    - HOLDQ_TERM_READ_RELQ_FAILED
- Signals from machines providing common services:
  - from "UPM_EXCEPT_RECOVERY_ACTION" on page 285
    - NOT_RETRIABLE
    - RETRIABLE_WITHOUT_MID_MU
    - RETRIABLE_RETRY_EXHAUSTED
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK
- Signals from LU 6.2 presentation services
  - OK
  - ALLOCATION_ERROR
  - DEALLOCATE_ABEND
  - PROG_ERROR
  - SVC_ERROR
  - RESOURCE_FAILURE

	States											
	RST	SEND ERR	EXCP ACT	HOLD SEND SIDE	DISC SEND SIDE	KEEP SEND SIDE	CONV FAIL	KEEP CONV FAIL	HOLD CONV FAIL	DISC CONV FAIL	PROG ERR	PROT ERR
Inputs	01	02	03	04	05	06	07	08	09	10	11	12
SEND SIDE EXCEPT	2a	/	/	/	/	/	/	/	/	/	/	/
ERROR NO DATA SENT	3b	/	/	/	/	/	/	/	/	/	/	/
CONV FAIL RCVD	7b	/	/	/	/	/	/	/	/	/	/	/
PROG ERR RCVD	11c	/	/	/	/	/	/	/	/	/	/	/
PROTO ERR RCVD	12c	/	/	/	/	/	/	/	/	/	/	/
OK	/	3b	/	/	/	/	/	/	/	/	/	/
ALLOCATION ERROR	/	7b	/	/	/	/	/	/	/	/	/	/
DEALLOCATE ABEND	/	7b	/	/	/	/	/	/	/	/	/	/
PROG ERROR	/	11c	/	/	/	/	/	/	/	/	/	/
SVC ERROR	/	12c	/	/	/	/	/	/	/	/	/	/
RESOURCE FAILURE	/	7b	/	/	/	/	/	/	/	/	/	/
NOT RETRIABLE	/	/	4i	/	/	/	9i	/	/	/	/	/
RETRIABLE WITHOUT MID MU	/	/	6d	/	/	/	8d	/	/	/	/	/
RETRIABLE RETRY EXHAUSTED	/	/	4i	/	/	/	9i	/	/	/	/	/
QUEUE OK	/	/	/	5c	/	/	/	/	10c	/	/	/
QUEUE NOT OK	/	/	/	5c	/	/	/	/	10c	/	/	/
DIST DISCARDED	/	/	/	/	1e	/	/	/	/	1f	1g	1h
QUEUE FAILED	/	/	/	/	1e	/	/	/	/	1f	1g	1h
SERVER FAILED	/	/	/	/	1e	/	/	/	/	1f	1g	1h
QUEUE AND SERVER FAILED	/	/	/	/	1e	/	/	/	/	1f	1g	1h
DIST RETAINED	/	/	/	/	/	1e	/	1f	/	/	/	/
RELQ FAILED	/	/	/	/	/	1e	/	1f	/	/	/	/
TERM READ FAILED	/	/	/	/	/	1e	/	1f	/	/	/	/
TERM READ RELQ FAILED	/	/	/	/	/	1e	/	1f	/	/	/	/
HOLDQ FAILED	/	/	/	/	/	1e	/	1f	/	/	/	/
HOLDQ RELQ FAILED	/	/	/	/	/	1e	/	1f	/	/	/	/
HOLDQ TERM READ FAILED	/	/	/	/	/	1e	/	1f	/	/	/	/
HOLDQ TERM READ RELQ FAILED	/	/	/	/	/	1e	/	1f	/	/	/	/

Output Code	Function
a	Signal LU 6.2 presentation services with SEND_ERROR.
b	Signal UPM_EXCEPT_RECOVERY_ACTION with the exception code.
c	Signal DS_SEND_DISCARD_DIST with START.
d	Signal DS_SEND_RETAIN_DIST with HOLDQ.
e	Signal caller with SEND_SIDE_EXCEPT.
f	Signal caller with CONVERSATION_FAILURE.
g	Signal caller with PROG_ERROR.
h	Signal caller with PROTOCOL_ERROR.
i	Signal QUEUE_MGR with HOLD to set an exception-hold for all next-DSU queues for this connection. No distributions will sent to the partner DSU until the cause of the exception is corrected. The control MU queue is not held.

DS_SEND_SEND_CONTROL_MU

Function: This finite-state machine describes the functional processing for sending a single control MU (PRMU, SEMU or CQMU). This routine reads the next CMU from the control MU queue, sends it, and discards it. If an exception occurs, the MU is not discarded, but is left on the queue to be resent later.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_SENDING" on page 156
    - SEND
- Signals from machines providing common services:
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK
    - QUEUE_EMPTY
  - from "IDLE_DETECTOR" on page 284
    - No signals are returned from this FSM.
  - from "DS_SEND_SEND_CONVERSATION_MGR" on page 220
    - OK
    - CONVERSATION_FAILURE
    - PROG_ERROR
    - PROTOCOL_ERROR

	States								
	RESET	SEND CMU	DEQ CMU	RET SENT	CONV FAIL	PROG ERR	RET SYS ERR	RET PROT ERR	RET CONV FAIL
Inputs	01	02	03	04	05	06	07	08	09
SEND	2a	/	/	/	/	/	/	/	/
QUEUE OK	/	3b	/	1g	1h	1i	1j	1k	1h
QUEUE NOT OK	/	7c	/	7c	9c	7c	1j	1k	1h
QUEUE EMPTY	/	1d	/	/	/	/	/	/	/
OK	/	/	4e	/	/	/	/	/	/
CONVERSATION FAILURE	/	/	5f	/	/	/	/	/	/
PROG ERROR	/	/	6f	/	/	/	/	/	/
PROTOCOL ERROR	/	/	8f	/	/	/	/	/	/

Output Code	Function
a	Signal QUEUE_MGR with READQ specifying *queue*(CONTROL_MU_QUEUE).
b	Signal IDLE_DETECTOR with SOMETHING_SENT. Signal DS_SEND_SEND_CONVERSATION_MGR with SEND_BUFFER to send the control MU to the partner DSU.
c	Notify operations of the exception condition. No DRMU will be generated because no distribution was involved. Signal QUEUE_MGR with HOLD to place an exception-hold on all next-DSU queues and also on the control MU queue for this connection. No more MUs will be sent to the partner DSU.
d	Signal caller with CONTROL_MU_QUEUE_EMPTY.
e	Signal QUEUE_MGR with DEQ, discarding the control MU.
f	Signal QUEUE_MGR with RELEASEQ, freeing the control MU for resending later.
g	Signal caller with CONTROL_MU_SENT.
h	Signal caller with CONVERSATION_FAILURE.
i	Signal caller with PROG_ERROR.
j	Signal caller with DS_SEND_SYSTEM_ERROR.
k	Notify operations of the exception condition. Signal caller with PROTOCOL_ERROR.

DS_SEND_RECEIVING

Function: This finite-state machine receives control MUs from the partner DSU. The FSM loops, receiving a CRMU or a REMU from the partner, parsing it and signalling the appropriate handler. The CRMU's *terminate_conversation* bit is also passed to DS_SEND_CONVERSATION_CONTROL. The FSM returns to the caller upon entering the LU 6.2 send state, detecting a conversation failure or deallocation, or encountering a violation of the DS usage of the LU 6.2 verbs. A PROG_ERROR indication (indicating that the partner issued a Send_Error verb) leaves the partner in send state and is simply ignored.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_MANAGER" on page 154
    - START
- Signals from lower-level DS_Send finite-state machines:
  - from "DS_SEND_REMU_HANDLER" on page 204
    - REMU_OK
    - DS_SEND_SYSTEM_ERROR
  - from "DS_SEND_CRMU_HANDLER" on page 192
    - CRMU_OK
    - DS_SEND_SYSTEM_ERROR
  - from "DS_SEND_CONVERSATION_CONTROL" on page 222
    - no signals are received from this FSM
- Signals from "PARSER" on page 359:
  - REMU
  - CRMU
  - PARSE_NOT_OK
- Signals from machines providing common services:
  - from "RCV_BUFFER_MGR" on page 282
    - OK
    - CONVERSATION_FAILURE
    - CHANGE_DIRECTION
    - PROG_ERROR
    - PROTOCOL_ERROR
    - DEALLOCATE_NORMAL
  - from "IDLE_DETECTOR" on page 284
    - no signals are received from this FSM

	States				
	RESET	RCV BUFFER	PARSE MU	HANDLE CRMU	HANDLE REMU
Inputs	01	02	03	04	05
START	2a	/	/	/	/
OK	/	3b	/	/	/
CONVERSATION FAILURE	/	1c	/	/	/
CHANGE DIRECTION	/	1d	/	/	/
PROG ERROR	/	-a	/	/	/
PROTOCOL ERROR	/	1e	/	/	/
DEALLOCATE NORMAL	/	1c	/	/	/
CRMU	/	/	4f	/	/
REMU	/	/	5g	/	/
PARSE NOT OK	/	/	1e	/	/
CRMU OK	/	/	/	2a	/
REMU OK	/	/	/	/	2a
DS SEND SYSTEM ERROR	/	/	/	1e	1e

Output Code	Function
a	Signal DS_SEND_CONVERSATION_CONTROL with RECEIVING. Signal RCV_BUFFER_MGR with RECEIVE_BUFFER.
b	Signal PARSER with PARSE.
c	Signal caller with DEALLOCATE_LOCAL.
d	Signal caller with CHANGE DIRECTION.
e	Notify operations of the exception condition. Signai caller with DEALLOCATE_ABEND.
f	Signal IDLE_DETECTOR with SOMETHING_RECEIVED. Signal DS_SEND_CONVERSATION_CONTROL with the *terminate_conversation* flag from the CRMU. Signal DS_SEND_CRMU_HANDLER with START.
g	Signal IDLE_DETECTOR with SOMETHING_RECEIVED. Signal DS_SEND_REMU_HANDLER with START.

DS_SEND_CRMU_HANDLER

Function: This finite-state machine processes a CRMU's *MU_ID state* indication, which may be COMPLETE, IN_TRANSIT, NOT_RECEIVED, TERMINATED, SUSPENDED, or PURGED.

In states 1-3, the FSM finds the specified distribution on the next-DSU queue, or discovers that the specified distribution does not exist. The queue hold indication may be ignored for this operation, since the hold is intended to prevent distributions from being sent, not to prevent CMU processing. CRMUs without *MU_ID*s are simply logged; those with old *instance numbers* are discarded.

In states 4-11, the specific actions taken for a given distribution and CRMU depend on the *MU_ID state*, whether the distribution can be retried (with a new *MU_ID* or with a DCMU), and the partner's state. The actions include purging the distribution because the partner has accepted responsibility, marking the distribution to be restarted later using a DCMU, purging the *MU_ID* and retrying later with a new *MU_ID*, and terminating the distribution. If the specified distribution does not exist and DS_Send's *MU_ID state* is PURGED, a PRMU is generated; if DS_Send's *MU_ID state* is NOT_ASSIGNED, the CRMU's *MU_ID state* is examined to determine if an MU sequence error has been detected. If so, an exception-hold is placed on the next-DSU queues.

Receiving a CRMU(IN_TRANSIT) may indicate a hung conversation, and should be reported to operations. At other times, the CRMU(IN_TRANSIT) probably indicates a race condition in which a SEMU was received before its accompanying Send_Error. In the latter cases, a CQMU is issued to solicit a CRMU(TERMINATED) or CRMU(SUSPENDED).

A CRMU(NOT_RECEIVED) may also indicate a hung conversation, or it may indicate a lost SEMU or DMU. A SEMU is perceived to be lost when an *MU_ID* with an *MU_ID state* of TERMINATION_PENDING or RETRY_PENDING is reported by the partner as NOT_RECEIVED; such a SEMU is simply reissued. A DMU is perceived to be lost when an *MU_ID* with an *MU_ID state* of CQMU_PENDING or TRANSFER_PENDING is reported by the partner as NOT_RECEIVED. Lost DMUs are handled by issuing a SEMU; the partner will respond to the SEMU with a CRMU(TERMINATED), which, in turn, solicits a PRMU.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_RECEIVING" on page 190
    - START

- Signals from lower-level DS_Send finite-state machines or procedures:
  - from “DS_SEND_MU_ID_REGISTRY” on page 223
    - NOT_ASSIGNED
    - TRANSF_PEND
    - CQMU_PEND
    - TERM_PEND
    - RETRY_PEND
    - SUSPENDED
    - INSTANCE_NUM_OK
    - INSTANCE_NUM_NOT_OK
    - MU_ID_OP_NOT_OK
  - from “DS_SEND_RELEASE_ON_CRMU” on page 196
    - CMU_OK
    - DS_SEND_SYSTEM_ERROR
  - from “DS_SEND_PURGE_ON_CRMU” on page 198
    - CMU_OK
    - DS_SEND_SYSTEM_ERROR
  - from “DS_SEND_RETRY_ON_CRMU” on page 200
    - CMU_OK
    - DS_SEND_SYSTEM_ERROR
  - from “DS_SEND_TERMINATE_DIST” on page 214
    - CMU_OK
    - DS_SEND_SYSTEM_ERROR
  - from “DS_SEND_ISSUE_SEMU_ON_CRMU” on page 202
    - CMU_OK
    - DS_SEND_SYSTEM_ERROR
- Signals from machines providing common services:
  - from “QUEUE_MGR” on page 357
    - QUEUE_OK
    - QUEUE_ENTRY_IN_USE
    - QUEUE_EMPTY
    - QUEUE_NOT_OK

	States												
	RST	FND DIS	INS NUM	INS REG	TRN PND	CQM PND	TRM PND	RTR PND	SUS	NOT FND	RET	NOT ASN	HLD
Inputs	01	02	03	04	05	06	07	08	09	10	11	12	13
START	2a	/	/	/	/	/	/	/	/	/	/	/	/
QUEUE OK	/	3b	/	/	/	/	/	/	/	/	/	/	1e
QUEUE ENTRY IN USE	/	1c	/	/	/	/	/	/	/	/	/	/	/
QUEUE EMPTY	/	10f	/	/	/	/	/	/	/	/	/	/	/
QUEUE NOT OK	/	1q	/	/	/	/	/	/	/	/	/	/	1q
INSTANCE NUM OK	/	/	4f	/	/	/	/	/	/	/	/	/	/
INSTANCE NUM NOT OK	/	/	11g	/	/	/	/	/	/	/	/	/	/
NOT ASSIGNED	/	/	/	/	/	/	/	/	/	12d	/	/	/
TRANSF PEND	/	/	/	5d	/	/	/	/	/	/	/	/	/
CQMU PEND	/	/	/	6d	/	/	/	/	/	/	/	/	/
TERM PEND	/	/	/	7d	/	/	/	/	/	/	/	/	/
RETRY PEND	/	/	/	8d	/	/	/	/	/	/	/	/	/
SUSPENDED	/	/	/	9d	/	/	/	/	/	/	/	/	/
PURGED	/	/	/	/	/	/	/	/	/	11p	/	/	/
MU ID OP NOT OK	/	/	11s	11s	/	/	/	/	/	/	/	/	/
CRMU(NOT RECEIVED)	/	/	/	/	11h	11m	11m	11m	/	/	/	1c	/
CRMU(IN TRANSIT)	/	/	/	/	11h	11h	11n	11n	/	/	/	13r	/
CRMU(SUSPENDED)	/	/	/	/	11i	11i	11o	11i	11i	/	/	13r	/
CRMU(TERMINATED)	/	/	/	/	11j	11j	11o	11j	11j	/	/	11p	/
CRMU(COMPLETED)	/	/	/	/	11k	11k	11k	11k	11k	/	/	13r	/
CRMU(PURGED)	/	/	/	/	/	/	/	/	/	/	/	13r	/
CMU OK	/	/	/	/	/	/	/	/	/	/	1c	/	/
DS SEND SYSTEM ERROR	/	/	/	/	/	/	/	/	/	/	1q	/	/

Output Code	**Function**
a	Signal QUEUE_MGR with READQ specifying *queue*(NEXT-DSU-QUEUE) to get the distribution specified by the CRMU. The *hold* flag may be ignored for this operation. If no *MU_ID* is specified in the CRMU, the CRMU is logged, reported to the operator, and processing continues as if QUEUE_MANAGER had returned QUEUE_ENTRY_IN_USE.
b	Signal DS_SEND_MU_ID_REGISTRY with INSTANCE_NUMBER to compare the CRMU's *instance number* with the current MU_ID registry *instance number*. If they do not match, then INSTANCE_NUM_NOT_OK is returned.
c	Signal caller with CRMU_OK.
d	Interrogate the *MU_ID state* of the CRMU.
e	Notify operations of the exception condition. Signal caller with CRMU_OK.
f	Signal DS_SEND_MU_ID_REGISTRY with INSPECT.
g	Signal DS_SEND_RELEASE_ON_CRMU with NO_ACTION.
h	Signal DS_SEND_RELEASE_ON_CRMU with NO_ACTION. Implementations may chose to notify operations that a hung conversation may exist.
i	Signal DS_SEND_RETRY_ON_CRMU with RETRY.
j	Signal DS_SEND_RETRY_ON_CRMU with RETRY_NO_MID_MU.
k	Signal DS_SEND_PURGE_ON_CRMU with DISCARD_AND_PURGE.
m	Signal DS_SEND_ISSUE_SEMU_ON_CRMU with START.
n	Signal DS_SEND_RELEASE_ON_CRMU with QUERY. This output code is usually used to handle a race condition in which a SEMU has outrun a Send_Error. However, in some circumstances, a hung conversation may exist, and implementations may notify the operator.
o	Signal DS_SEND_TERMINATE_DIST with REPORT_AND_PURGE.
p	Signal DS_SEND_PURGE_ON_CRMU with PURGE.
q	Signal caller with DS_SEND_SYSTEM_ERROR.
r	An MU sequence error has been detected. Signal QUEUE_MGR with HOLD to place an exception-hold on all next-DSU queues.
s	Signal DS_SEND_RELEASE_ON_CRMU with MU_ID_EXCEPT.

DS_SEND_RELEASE_ON_CRMU

Function: This finite-state machine is signalled when no processing is to be performed on the distribution or *MU_ID*. It is signalled in the following cases:

- The CRMU's *instance number* is low, and the CRMU is to be discarded.
- An MU_ID Registry exception has occurred.
- The partner DSU is perceived to be impatient, and has sent a CRMU(NOT_RECEIVED) for an *MU_ID* that is either NOT_ASSIGNED or IN_TRANSIT.
- The partner DSU is perceived to be impatient, and reports an *MU_ID* as IN_TRANSIT when DS_Send expects some other active state.

A CQMU is issued, if requested.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_CRMU_HANDLER" on page 192
    - QUERY
    - NO_ACTION
    - MU_ID_EXCEPT
- Signals from machines providing common services:
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK

Inputs	States			
	RESET	BUILD CQMU	RELQ	MU_ID EXCEPT
	01	02	03	04
QUERY	2a	/	/	/
NO ACTION	3b	/	/	/
MU ID EXCEPT	4b	/	/	/
QUEUE OK	/	3b	1d	1c
QUEUE NOT OK	/	1c	1e	1c

Output Code	Function
a	Build the CQMU and signal QUEUE_MGR with WRITEQ specifying *queue*(CONTROL_MU_QUEUE) to determine the restart position.
b	Notify operations of the exception condition. Signal QUEUE_MGR with RELEASEQ to remove the in-use mark from the entry on the next-DSU queue. Following the RELEASEQ, the entry will be available for processing.
c	Signal caller with DS_SEND_SYSTEM_ERROR.
d	Signal caller with CMU_OK.
e	Notify operations of the exception condition. Signal caller with CMU_OK.

DS_SEND_PURGE_ON_CRMU

Function: This finite-state machine changes the *MU_ID state* to PURGED, and issues a PRMU to notify the partner that the *MU_ID* has been purged. It optionally discards the distribution. This FSM is called whenever the partner has taken responsibility for the distribution by issuing a CRMU(COMPLETE), or when the partner reports as active an *MU_ID* that DS_Send has purged.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_CRMU_HANDLER" on page 192
    - PURGE
    - DISCARD_AND_PURGE
- Signals from lower-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_DISCARD_DIST" on page 218
    - DIST_DISCARDED
    - QUEUE_FAILED
    - SERVER_FAILED
    - QUEUE_AND_SERVER_FAILED
  - from "DS_SEND_MU_ID_REGISTRY" on page 223
    - MU_ID_OP_OK
    - MU_ID_OP_NOT_OK
- Signals from machines providing common services:
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK

	States				
	RESET	**PURGE MU_ID**	**DISC DIST**	**DISC PURGE MU_ID**	**PRMU**
Inputs	**01**	**02**	**03**	**04**	**05**
PURGE	2a	/	/	/	/
DISCARD AND PURGE	3e	/	/	/	/
MU ID OP OK	/	5c	/	5c	/
MU ID OP NOT OK	/	1d	/	1d	/
QUEUE OK	/	/	/	/	1b
QUEUE NOT OK	/	/	/	/	1d
DIST DISCARDED	/	/	4a	/	/
QUEUE FAILED	/	/	1f	/	/
SERVER FAILED	/	/	4a	/	/
QUEUE AND SERVER FAILED	/	/	1f	/	/

Output Code	**Function**
a	Signal DS_SEND_MU_ID_REGISTRY with PURGE. If the *MU_ID* has already been purged, DS_SEND_MU_ID_REGISTRY returns MU_ID_OP_OK.
b	Signal caller with CMU_OK.
c	Build the PRMU and signal QUEUE_MGR with WRITEQ specifying *queue*(CONTROL_MU_QUEUE) to purge the partner's knowledge of the *MU_ID*.
d	Signal caller with DS_SEND_SYSTEM_ERROR.
e	Signal DS_SEND_DISCARD_DIST with START.
f	Notify operations of the exception condition. Signal caller with CMU_OK.

DS_SEND_RETRY_ON_CRMU

Function: If a distribution's transfer to the partner is interrupted by an exception, DS_Send determines whether to terminate the distribution or to attempt to recover from the exception. If DS_Send decides to recover, this FSM is called. It either suspends the distribution by setting the *MU_ID state* to SUSPENDED, or it purges the *MU_ID* and prepares the distribution to be retried later with a new *MU_ID*.

Suspending the distribution merely involves setting the *MU_ID state* to SUSPENDED, recording the appropriate restart position for the DCMU, and issuing a RELEASEQ signal to the queue manager. Depending on implementation-defined circumstances (e.g., the exception may have been caused by an operator suspending this transmission), implementations may put special hold conditions on the distribution.

Purging the *MU_ID* involves setting the *MU_ID state* to PURGED and issuing a PRMU to inform the partner. Preparing the distribution to be retried with a new *MU_ID* involves terminating the server's restartability (if appropriate), and issuing the RELEASEQ signal to the queue manager.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_CRMU_HANDLER" on page 192
    - RETRY
    - RETRY_NO_MID_MU
- Signals from lower-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_MU_ID_REGISTRY" on page 223
    - MU_ID_OP_OK
    - MU_ID_OP_NOT_OK
  - from "UPM_EXCEPT_RECOVERY_ACTION" on page 285
    - RETRIABLE_WITH_MID_MU
    - RETRIABLE_WITHOUT_MID_MU
- Signals from machines providing common services:
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK
  - from "SERVER_MGR" on page 354
    - OBJECT_OK
    - OBJECT_NOT_OK

	States								
	RESET	SUSP	SUSP RELQ	PURGE MU_ID	TERM REST	NOT ASSIGN	PRMU	SYS ERR RELQ	RELQ
Inputs	01	02	03	04	05	06	07	08	09
RETRY	2a	/	/	/	/	/	/	/	/
RETRY NO MID MU	4b	/	/	/	/	/	/	/	/
RETRIABLE WITH MID MU	/	3c	/	/	/	/	/	/	/
RETRIABLE WITHOUT MID MU	/	4b	/	/	/	/	/	/	/
MU ID OP OK	/	/	7d	/	6f	/	/	/	/
MU ID OP NOT OK	/	/	8d	/	8d	/	/	/	/
OBJECT OK	/	/	/	5e	/	/	/	/	/
OBJECT NOT OK	/	/	/	5e	/	/	/	/	/
QUEUE OK	/	/	/	/	/	9d	1g	1h	1g
QUEUE NOT OK	/	/	/	/	/	8d	1g	1h	1h

Output Code	Function
a	Signal UPM_EXCEPT_RECOVERY_ACTION with the receiver's restart position.
b	Signal SERVER_MGR with TERMINATE_RESTARTABILITY.
c	Signal DS_SEND_MU_ID_REGISTRY with SUSPENDED.
d	Notify operations of the exception condition. Signal QUEUE_MGR with RELEASEQ to remove the in-use mark from the entry on the next-DSU queue. Depending on the circumstances surrounding the distribution's continuation (for example, the distribution's transfer may have been suspended by an operator), implementations may mark the distribution with an operator-hold. An operator-hold must be explicitly released by the operator before it can be continued. If no operator-hold is placed on the distribution, then following the RELEASEQ it will be available for processing.
e	Signal DS_SEND_MU_ID_REGISTRY with PURGE.
f	Build the PRMU and signal QUEUE_MGR with WRITEQ specifying *queue*(CONTROL_MU_QUEUE) to purge the *MU_ID* value in the partner DSU.
g	Notify operations of the exception condition. Signal caller with CMU_OK.
h	Notify operations of the exception condition. Signal caller with DS_SEND_SYSTEM_ERROR.

DS_SEND_ISSUE_SEMU_ON_CRMU

Function:	This finite-state machine issues a SEMU and releases the distribution for later processing. The SEMU may have been lost (or at least appears to have been lost), or may be issued to terminate an apparently lost distribution. This FSM gets control from one of the following: • Signals from higher-level DS_Send finite-state machines or procedures: – from "DS_SEND_CRMU_HANDLER" on page 192 — START • Signals from machines providing common services: – from "QUEUE_MGR" on page 357 — QUEUE_OK — QUEUE_NOT_OK

	States		
	RESET	**SEMU**	**RELQ**
Inputs	**01**	**02**	**03**
START	2a	/	/
QUEUE OK	/	3b	1d
QUEUE NOT OK	/	1c	1e

Output Code	**Function**
a	If a SEMU has been previously issued for this distribution, then rebuild an exact copy of the original SEMU and signal QUEUE_MGR with WRITEQ specifying *queue*(CONTROL_MU_QUEUE) to send it. If a SEMU has not been issued previously, or no such distribution exists, then build a SEMU using SNA Report Code "System Exception--Identifiable" and signal QUEUE_MGR with WRITEQ specifying *queue*(CONTROL_MU_QUEUE) to send it.
b	Notify operations of the exception condition. Signal QUEUE_MGR with RELEASEQ to remove the in-use mark from the entry on the next-DSU queue. Following the RELEASEQ, the entry will be available for processing.
c	Signal caller with DS_SEND_SYSTEM_ERROR.
d	Signal caller with CMU_OK.
e	Notify operations of the exception condition. Signal caller with CMU_OK.

DS_SEND_REMU_HANDLER

Function: This finite-state machine handles REMUs. In states 1-3, the FSM finds the specified distribution on the next-DSU queue. The queue hold indication is ignored for this operation, since the hold is intended to prevent more distributions from being sent, not to prevent control processing of distributions with active *MU_ID states*. If no *MU_ID* is specified, the REMU is logged, but no other action is taken. If the specified distribution does not exist and the *MU_ID state* is PURGED, the REMU is discarded; if the *MU_ID state* is NOT_ASSIGNED, the partner has committed an MU sequence error, and an exception-hold is placed on the next-DSU queues. If the *instance number* given in the REMU does not match that in the distribution, the REMU is discarded. REMUs reporting already-detected conversation failures are discarded, since the recovery processing for these failures has already been initiated by a SEMU.

In states 4-10, the specific actions taken for a given distribution and REMU depend on the *MU_ID state*, the ability of the DSU to retry the distribution with a new *MU_ID* or restart it with a DCMU, and the REMU′s exception condition. The actions include querying the partner to determine the restart position, purging the *MU_ID* and retrying the distribution with a new *MU_ID* at a later time, and terminating the distribution. Retriable exceptions always result in the next-DSU queues being held.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from “DS_SEND_RECEIVING” on page 190
    - START
- Signals from lower-level DS_Send finite-state machines:
  - from “DS_SEND_MU_ID_REGISTRY” on page 223
    - TRANSF_PEND
    - CQMU_PEND
    - TERM_PEND
    - RETRY_PEND
    - INSTANCE_NUM_OK
    - INSTANCE_NUM_NOT_OK
    - MU_ID_OP_NOT_OK
  - from “DS_SEND_QUERY_ON_REMU” on page 208
    - CMU_OK
    - DS_SEND_SYSTEM_ERROR
  - from “DS_SEND_RETRY_ON_REMU” on page 210
    - CMU_OK
    - DS_SEND_SYSTEM_ERROR
  - from “DS_SEND_TERMINATE_DIST” on page 214
    - CMU_OK
    - DS_SEND_SYSTEM_ERROR
  - from “DS_SEND_CHECK_CONV_FAIL” on page 212
    - NOT_RETRIABLE
    - RETRIABLE_WITH_MID_MU
    - RETRIABLE_WITHOUT_MID_MU
    - RETRIABLE_RETRY_EXHAUSTED
    - DUP_CONV_FAIL_REPORT

- Signals from machines providing common services:
  - from “UPM_EXCEPT_RECOVERY_ACTION” on page 285
    - NOT_RETRIABLE
    - RETRIABLE_WITH_MID_MU
    - RETRIABLE_WITHOUT_MID_MU
    - RETRIABLE_RETRY_EXHAUSTED
  - from “QUEUE_MGR” on page 357
    - QUEUE_OK
    - QUEUE_EMPTY
    - QUEUE_ENTRY_IN_USE
    - QUEUE_NOT_OK

	States												
	RST	READ	INST NUM	MUID STAT	TRAN PEND	CQ-MU PEND	TERM PEND	RTRY PEND	RET	RELQ	SYS ERR	NO DIS	MU SEQ ERR
Inputs	01	02	03	04	05	06	07	08	09	10	11	12	13
START	2a	/	/	/	/	/	/	/	/	/	/	/	/
QUEUE OK	/	3b	/	/	/	/	/	/	/	1c	1n	/	1p
QUEUE EMPTY	/	12e	/	/	/	/	/	/	/	/	/	/	/
QUEUE ENTRY IN USE	/	-d	/	/	/	/	/	/	/	/	/	/	/
QUEUE NOT OK	/	1n	/	/	/	/	/	/	/	1c	1n	/	1p
INSTANCE NUM OK	/	/	4e	/	/	/	/	/	/	/	/	/	/
INSTANCE NUM NOT OK	/	/	10f	/	/	/	/	/	/	/	/	/	/
NOT STARTED	/	/	/	/	/	/	/	/	/	/	/	13o	/
TRANSF PEND	/	/	/	5g	/	/	/	/	/	/	/	/	/
CQMU PEND	/	/	/	6g	/	/	/	/	/	/	/	/	/
TERM PEND	/	/	/	7h	/	/	/	/	/	/	/	/	/
RETRY PEND	/	/	/	8h	/	/	/	/	/	/	/	/	/
SUSPENDED	/	/	/	10f	/	/	/	/	/	/	/	/	/
PURGED	/	/	/	/	/	/	/	/	/	/	/	1c	/
MU ID OP NOT OK	/	/	11f	11f	/	/	/	/	/	/	/	/	/
NOT RETRIABLE	/	/	/	/	9i	9i	9i	9i	/	/	/	/	/
RETRIABLE WITH MID MU	/	/	/	/	9j	10f	9m	9j	/	/	/	/	/
RETRIABLE WITHOUT MID MU	/	/	/	/	9k	9k	9m	9k	/	/	/	/	/
RETRIABLE RETRY EXHAUSTED	/	/	/	/	9m	9m	9m	9m	/	/	/	/	/
CMU OK	/	/	/	/	/	/	/	/	1c	/	/	/	/
DS SEND SYSTEM ERROR	/	/	/	/	/	/	/	/	1n	/	/	/	/
DUP CONV FAIL REPORT	/	/	/	/	/	/	1c	1c	/	/	/	/	/

Output Code	Function
a	Signal QUEUE_MANAGER with READQ specifying *queue*(NEXT-DSU-QUEUE) to get the distribution specified in the REMU. The HOLD indication is ignored for this operation. If no *MU_ID* is specified in the REMU, the REMU is logged, reported to the operator, and processing continues as if QUEUE_MANAGER had returned QUEUE_EMPTY and the *MU_ID state* was PURGED.
b	Signal DS_SEND_MU_ID_REGISTRY with INSTANCE_NUMBER to compare the REMU's *instance number* with the the current MU_ID registry *instance number*. If they do not match, then INSTANCE_NUM_NOT_OK is returned.
c	Signal caller with REMU_OK.
d	Signal QUEUE_MANAGER with READQ specifying *queue*(NEXT-DSU-QUEUE) *suspend* to get the distribution. Implementations may set a timer to avoid suspending for too long. If the timer expires before the distribution is available, the operator is notified that a potential hung session exists, and processing continues as though QUEUE_EMPTY had been returned and the *MU_ID state* were PURGED. Implementations unable to suspend for the READQ may set a timer and retry the READQ when the timer expires. If this subsequent READQ fails, the operator is notified and processing continues as though QUEUE_EMPTY had been returned and the *MU_ID state* were PURGED.
e	Signal DS_SEND_MU_ID_REGISTRY with INSPECT.
f	Signal QUEUE_MGR with RELEASEQ to remove the in-use mark from the entry on the next-DSU queue. Following the RELEASEQ, the entry will be available for processing.
g	Signal UPM_EXCEPT_RECOVERY_ACTION with the relevant fields from the REMU.
h	Signal DS_SEND_CHECK_CONV_FAIL with START.
i	Signal DS_SEND_TERMINATE_DIST with REPORT_AND_PURGE.
j	Signal DS_SEND_QUERY_ON_REMU with START.
k	Signal DS_SEND_RETRY_ON_REMU with START.
m	Signal DS_SEND_TERMINATE_DIST with HOLD_AND_REPORT_AND_PURGE.
n	Signal caller with DS_SEND_SYSTEM_ERROR.
o	An MU sequence error has been detected. Signal QUEUE_MGR with HOLD to hold all next-DSU queues.
p	Notify operations of the exception condition. Signal caller with REMU_OK.

DS_SEND_QUERY_ON_REMU

Function: This finite-state machine retains the distribution (i.e., Terminate_Read server verbs and the RELEASEQ queue operation are performed), holds the next-DSU queues with an exception-hold, sets the *MU_ID state* to CQMU_PENDING, and issues a CQMU to query the partner. This FSM is signalled when a REMU reporting a retriable exception is received, mid-MU restart is possible, and the sender's *MU_ID state* is TRANSFER_PENDING or RETRY_PENDING.

Failures involving the MU_ID registry or the control MU queue cause DS_SEND_SYSTEM_ERROR to be returned. Otherwise, CMU_OK is returned.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_REMU_HANDLER" on page 204
    - START
- Signals from lower-level DS_Send finite-state machines:
  - from "DS_SEND_RETAIN_DIST" on page 216
    - DIST_RETAINED
    - RELQ_FAILED
    - TERM_READ_FAILED
    - TERM_READ_RELQ_FAILED
    - HOLDQ_FAILED
    - HOLDQ_RELQ_FAILED
    - HOLDQ_TERM_READ_FAILED
    - HOLDQ_TERM_READ_RELQ_FAILED
  - from "DS_SEND_MU_ID_REGISTRY" on page 223
    - MU_ID_OP_OK
    - MU_ID_OP_NOT_OK
- Signals from machines providing common services:
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK

Inputs	States			
	RESET	CQMU PEND	RETAIN DIST	CQMU
	01	02	03	04
START	2a	/	/	/
DIST RETAINED	/	3b	/	/
RELQ FAILED	/	3b	/	/
TERM READ FAILED	/	3b	/	/
TERM READ RELQ FAILED	/	3b	/	/
HOLDQ FAILED	/	3b	/	/
HOLDQ RELQ FAILED	/	3b	/	/
HOLDQ TERM READ FAILED	/	3b	/	/
HOLDQ TERM READ RELQ FAILED	/	3b	/	/
MU ID OP OK	/	/	4c	/
MU ID OP NOT OK	/	/	1e	/
QUEUE OK	/	/	/	1d
QUEUE NOT OK	/	/	/	1e

Output Code	Function
a	Signal DS_SEND_RETAIN_DIST with HOLDQ.
b	Signal DS_SEND_MU_ID_REGISTRY with CQMU_PENDING to set the *MU_ID state*.
c	Build the CQMU and signal QUEUE_MGR with WRITEQ specifying *queue*(CONTROL_MU_QUEUE) to determine the restart position. If the sender had detected an exception previously and issued a SEMU, implementations may choose to suppress this CQMU, since the SEMU will solicit a CRMU. In such cases, processing continues as though QUEUE_OK had been returned.
d	Notify operations of the exception condition. Signal caller with CMU_OK.
e	Notify operations of the exception condition. Signal caller with DS_SEND_SYSTEM_ERROR.

DS_SEND_RETRY_ON_REMU

Function: This FSM prepares a distribution for retry with a new *MU_ID*. The (existing) *MU_ID* is purged from the registry, a PRMU is sent, the next-DSU queues are held with an exception-hold, and the *MU_ID state* is changed to NOT_ASSIGNED. A subsequent invocation of DS_Send will assign a new *MU_ID* to this distribution and resend it.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_REMU_HANDLER" on page 204
    - START
- Signals from lower-level DS_Send finite-state machines:
  - from "DS_SEND_RETAIN_DIST" on page 216
    - DIST_RETAINED
    - RELQ_FAILED
    - TERM_READ_FAILED
    - TERM_READ_RELQ_FAILED
    - HOLDQ_FAILED
    - HOLDQ_RELQ_FAILED
    - HOLDQ_TERM_READ_FAILED
    - HOLDQ_TERM_READ_RELQ_FAILED
  - from "DS_SEND_MU_ID_REGISTRY" on page 223
    - MU_ID_OP_OK
    - MU_ID_OP_NOT_OK
- Signals from machines providing common services:
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK
  - from "SERVER_MGR" on page 354
    - OBJECT_OK
    - OBJECT_NOT_OK

	States				
	RESET	PURGE MU_ID	TERM RESTART	RETAIN DIST	PRMU
Inputs	01	02	03	04	05
START	2a	/	/	/	/
MU ID OP OK	/	3b	/	/	/
MU ID OP NOT OK	/	1c	/	/	/
OBJECT OK	/	/	4d	/	/
OBJECT NOT OK	/	/	4d	/	/
DIST RETAINED	/	/	/	5e	/
RELQ FAILED	/	/	/	5e	/
TERM READ FAILED	/	/	/	5e	/
TERM READ RELQ FAILED	/	/	/	5e	/
HOLDQ FAILED	/	/	/	5e	/
HOLDQ RELQ FAILED	/	/	/	5e	/
HOLDQ TERM READ FAILED	/	/	/	5e	/
HOLDQ TERM READ RELQ FAILED	/	/	/	5e	/
QUEUE OK	/	/	/	/	1f
QUEUE NOT OK	/	/	/	/	1c

Output Code	Function
a	Signal DS_SEND_MU_ID_REGISTRY with PURGE.
b	Signal SERVER_MGR with TERMINATE_RESTARTABILITY, if applicable.
c	Notify operations of the exception condition. Signal caller with DS_SEND_SYSTEM_ERROR.
d	Signal DS_SEND_RETAIN_DIST with HOLDQ.
e	Build the PRMU and signal QUEUE_MGR with WRITEQ specifying *queue*(CONTROL_MU_QUEUE) to purge the partner's knowledge of the *MU_ID*.
f	Notify operations of the exception condition. Signal caller with CMU_OK.

DS_SEND_CHECK_CONV_FAIL

Function: This finite-state machine determines if the just-received REMU is reporting an already-detected conversation failure. If so, the REMU will be discarded. If not, this routine returns the appropriate retry action. This FSM is signalled whenever a REMU is received for an *MU_ID state* of TERMINATION_PENDING or RETRY_PENDING.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_REMU_HANDLER" on page 204.
    - START
- Signals from lower-level DS_Send finite-state machines:
  - from "UPM_CHECK_DUP_CONV_FAIL_REPORT" on page 224
    - DUP_CONV_FAIL_REPORT
    - NOT_DUP_CONV_FAIL_REPORT
- Signals from machines providing common services:
  - from "UPM_EXCEPT_RECOVERY_ACTION" on page 285
    - NOT_RETRIABLE
    - RETRIABLE_WITH_MID_MU
    - RETRIABLE_WITHOUT_MID_MU
    - RETRIABLE_RETRY_EXHAUSTED
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK

	States			
	RESET	CONV FAIL CHECK	RELQ	RETRY ACTION
Inputs	01	02	03	04
START	2a	/	/	/
DUP CONV FAIL REPORT	/	3b	/	/
NOT DUP CONV FAIL REPORT	/	4c	/	/
QUEUE OK	/	/	1d	/
QUEUE NOT OK	/	/	1d	/
NOT RETRIABLE	/	/	/	1e
RETRIABLE WITH MID MU	/	/	/	1f
RETRIABLE WITHOUT MID MU	/	/	/	1g
RETRIABLE RETRY EXHAUSTED	/	/	/	1h

Output Code	Function
a	Signal UPM_CHECK_DUP_CONV_FAIL_REPORT with START. If DS_Send detected a conversation failure on this distribution and sent a SEMU reporting conversation failure, and if the just-received REMU reports conversation failure, then the UPM returns DUP_CONV_FAIL_REPORT. Otherwise, NOT_DUP_CONV_FAIL_REPORT is returned.
b	Signal QUEUE_MGR with RELEASEQ to remove the in-use mark from the entry on the next-DSU queue. Following the RELEASEQ, the entry will be available for processing.
c	Signal UPM_EXCEPT_RECOVERY_ACTION with the relevant fields from the REMU.
d	Signal caller with DUP_CONV_FAIL_REPORT.
e	Signal caller with NOT_RETRIABLE.
f	Signal caller with RETRIABLE_WITH_MID_MU.
g	Signal caller with RETRIABLE_WITHOUT_MID_MU.
h	Signal caller with RETRIABLE_RETRY_EXHAUSTED.

DS_SEND_TERMINATE_DIST

Function: This finite-state machine is called when a REMU or CRMU is received and DS_Send determines that an exception condition makes reattempting transmission of the distribution inappropriate. Terminating a distribution involves

- Holding the next-DSU queues (if requested to do so).
- Generating a distribution report (if appropriate).
- Setting the *MU_ID state* to PURGED.
- Signalling DS_SEND_DISCARD_DIST to
  - Remove the distribution from the next-DSU queue and discard it.
  - Terminate the server's restartability for the server object (if appropriate).
  - Decrement the server's DS lock count for the server object (if appropriate).
- generating a PRMU to the partner.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_REMU_HANDLER" on page 204
    - HOLD_AND_REPORT_AND_PURGE
    - REPORT_AND_PURGE
  - from "DS_SEND_CRMU_HANDLER" on page 192
    - REPORT_AND_PURGE
- Signals from lower-level DS_Send finite-state machines:
  - from "DS_SEND_DISCARD_DIST" on page 218
    - DIST_DISCARDED
    - QUEUE_FAILED
    - SERVER__FAILED
    - QUEUE_AND_SERVER_FAILED
  - from "DS_SEND_MU_ID_REGISTRY" on page 223
    - MU_ID_OP_OK
    - MU_ID_OP_NOT_OK
- Signals from machines providing common services:
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK

Inputs	States					
	RESET	HOLDQ	DRMU	PURGE MU_ID	DISC DIST	PRMU
	01	02	03	04	05	06
HOLD AND REPORT AND PURGE	2a	/	/	/	/	/
REPORT AND PURGE	3b	/	/	/	/	/
QUEUE OK	/	3b	4c	/	/	1g
QUEUE NOT OK	/	3b	4c	/	/	1e
MU ID OP OK	/	/	/	5d	/	/
MU ID OP NOT OK	/	/	/	1e	/	/
DIST DISCARDED	/	/	/	/	6f	/
QUEUE FAILED	/	/	/	/	1g	/
SERVER FAILED	/	/	/	/	6f	/
QUEUE AND SERVER FAILED	/	/	/	/	1g	/

Output Code	Function
a	Signal QUEUE_MGR with HOLD specifying an exception-hold for all next-DSU queues for this connection. No distributions should be sent to the partner DSU until the cause of the exception is corrected. The control MU queue is not held.
b	Generate the distribution report, if appropriate, and signal QUEUE_MGR with WRITEQ specifying *queue*(ROUTER_DIRECTOR_QUEUE). If a report is not appropriate, QUEUE_MGR returns QUEUE_OK.
c	Signal DS_SEND_MU_ID_REGISTRY with PURGE.
d	Signal DS_SEND_DISCARD_DIST with START.
e	Notify operations of the exception condition. Signal caller with DS_SEND_SYSTEM_ERROR.
f	Build the PRMU and signal QUEUE_MGR with WRITEQ specifying *queue*(CONTROL_MU_QUEUE) to purge the partner's knowledge of the MU_ID.
g	Notify operations of the exception condition. Signal caller with CMU_OK.

DS_SEND_RETAIN_DIST

Function: When processing on a distribution is halted temporarily, this FSM is signalled to terminate any server operations in progress and to clear the distribution's in-use mark, making the distribution available for further processing. Retaining a distribution involves the following actions:

- The next-DSU queues are held with an exception-hold, if requested.
- A Terminate_Read verb is issued to the server, but the restartability of the server (if originally requested) is maintained.
- A RELEASEQ signal is sent to queue manager to remove the in-use mark from the distribution. The distribution will then be available to other DS_Send processes.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_CLEANUP_DMU" on page 166.
    - NO_HOLDQ
  - from "DS_SEND_CLEANUP_EXCEPT" on page 172.
    - HOLDQ
    - NO_HOLDQ
  - from "DS_SEND_PROG_ERROR_RECEIVED" on page 174.
    - NO_HOLDQ
  - from "DS_SEND_DMU_PROTOCOL_ERROR" on page 176.
    - HOLDQ
  - from "DS_SEND_EXCEPT_NO_MU_ID" on page 185.
    - HOLDQ
  - from "DS_SEND_QUERY_ON_REMU" on page 208.
    - HOLDQ
  - from "DS_SEND_RETRY_ON_REMU" on page 210.
    - HOLDQ
    - NO_HOLDQ
- Signals from machines providing common services:
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK
  - from "SERVER_MGR" on page 354
    - OBJECT_OK
    - OBJECT_NOT_OK

	States							
	RESET	HOLDQ	TERM READ	RELQ	TERM FAIL RELQ	HOLD FAIL TERM	HOLD FAIL RELQ	BOTH FAIL RELQ
Inputs	01	02	03	04	05	06	07	08
HOLDQ	2a	/	/	/	/	/	/	/
NO HOLDQ	3b	/	/	/	/	/	/	/
QUEUE OK	/	3b	/	1d	1f	/	1h	1j
QUEUE NOT OK	/	6b	/	1e	1g	/	1i	1k
OBJECT OK	/	/	4c	/	/	7c	/	/
OBJECT NOT OK	/	/	5c	/	/	8c	/	/

Output Code	Function
a	Signal QUEUE_MGR with HOLD specifying an exception-hold on all next-DSU queues for this connection; no more distributions should be sent to the partner DSU. The control MU queue is not held.
b	Signal SERVER_MGR with TERMINATE_READ specifying SUSPEND. If no Initiate_Read has opened the server object or no server object exists for this distribution, SERVER_MGR returns OBJECT_OK.
c	Signal QUEUE_MGR with RELEASEQ to remove the in-use mark from the entry on the next-DSU queue. Following the RELEASEQ, the entry will be available for processing.
d	Signal caller with DIST_RETAINED.
e	Notify operations of the exception condition. Signal caller with RELQ_FAILED.
f	Notify operations of the exception condition. Signal caller with TERM_READ_FAILED.
g	Notify operations of the exception condition. Signal caller with TERM_READ_RELQ_FAILED.
h	Notify operations of the exception condition. Signal caller with HOLDQ_FAILED.
i	Notify operations of the exception condition. Signal caller with HOLDQ_RELQ_FAILED.
j	Notify operations of the exception condition. Signal caller with HOLDQ_TERM_READ_FAILED.
k	Notify operations of the exception condition. Signal caller with HOLDQ_TERM_READ_RELQ_FAILED.

DS_SEND_DISCARD_DIST

Function: This finite-state machine describes the functional processing for deleting a distribution from the DSU. This involves the following actions:

- If a server object exists for this distribution, it is deleted. This may involve:
  - A Terminate_Read or Terminate_Restartability server verb is issued, if appropriate.
  - The DS lock count for the server object is decremented by the server manager, and the server object may be deleted from non-volatile storage if both the agent and DS lock counts are 0.
- The queue manager is signalled with DEQ to remove the distribution from the next-DSU queue.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_SEND_DMU_NO_MU_ID" on page 181
    - START
  - from "DS_SEND_EXCEPT_NO_MU_ID" on page 185
    - START
  - from "DS_SEND_PURGE_ON_CRMU" on page 198
    - START
  - from "DS_SEND_TERMINATE_DIST" on page 214
    - START
- Signals from machines providing common services:
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK
  - from "SERVER_MGR" on page 354
    - OBJECT_OK
    - OBJECT_NOT_OK

	States					
	RESET	TERM	DEC	DEQ	FAIL DEC	FAIL DEQ
Inputs	01	02	03	04	05	06
START	2a	/	/	/	/	/
OBJECT OK	/	3b	4c	/	6c	/
OBJECT NOT OK	/	5b	6c	/	6c	/
QUEUE OK	/	/	/	1d	/	1f
QUEUE NOT OK	/	/	/	1e	/	1g

Output Code	Function
a	If appropriate, signal SERVER_MGR with TERMINATE_READ terminating restartability of the server object. If a Terminate_Read specifying SUSPEND has been issued previously, then signal SERVER_MGR with TERMINATE_RESTARTABILITY, if appropriate. If no Initiate_Read has opened the server object or no server object exists for this distribution, SERVER_MGR returns OBJECT_OK.
b	Signal SERVER_MGR with DECREMENT_OBJ_LOCK to decrement the DS lock count on the server object. If no server object exists, OBJECT_OK is returned.
c	Signal QUEUE_MGR with DEQ to remove the distribution from the next-DSU queue.
d	Signal caller with DIST_DISCARDED.
e	Notify operations of the exception condition. Signal caller with QUEUE_FAILED.
f	Notify operations of the exception condition. Signal caller with SERVER_FAILED.
g	Notify operations of the exception condition. Signal caller with QUEUE_AND_SERVER_FAILED.

DS_SEND_SEND_CONVERSATION_MGR

Function: This finite-state machine describes the functional processing for sending a buffer to LU 6.2 presentation services using the the Send_Data verb.

This FSM gets control from one of the following:

- Signals from higher-level finite-state machines:
  - from "DS_SEND_BUILD_SEND_DMU" on page 163
    - SEND_BUFFER
  - from "DS_SEND_SEND_DMU_NO_MU_ID" on page 181
    - SEND_BUFFER
  - from "DS_SEND_SEND_CONTROL_MU" on page 188
    - SEND_BUFFER
- Signals from lower-level finite-state machines:
  - from "DS_SEND_CONVERSATION_CONTROL" on page 222
    - no signals received from this FSM.
- Signals from LU 6.2 presentation services:
  - ALLOCATION_ERROR
  - DEALLOCATE_ABEND
  - PROG_ERROR
  - SVC_ERROR
  - RESOURCE_FAILURE
  - OK

	States	
	RESET	SEND DATA
Inputs	01	02
SEND BUFFER	2a	/
OK	/	1b
DEALLOCATE ABEND	/	1c
RESOURCE FAILURE	/	1c
PROG ERROR	/	1d
SVC ERROR	/	1e
ALLOCATION ERROR	/	1c

Output Code	Function
a	Signal LU 6.2 presentation services with Send_Data to send the encoded information to the partner DSU.
b	Signal caller with OK to indicate that the LU 6.2 request was completed successfully.
c	Signal caller with CONVERSATION_FAILURE to indicate that some error has occurred on the conversation between DS_Send and DS_Receive.
d	Signal caller with PROG_ERROR to indicate that the partner DSU has signalled that an error has occurred.
e	Signal caller with PROTOCOL_ERROR to indicate a violation of DS use of the LU 6.2 basic conversation verbs has occurred between DS_Send and DS_Receive.

DS_SEND_CONVERSATION_CONTROL

Function: This finite-state machine remembers if DS_Send is allowed to send a distribution, send only control MUs, or must terminate the conversation. OPERATOR_QUIESCE_CONVERSATION shows the actions for handling an operator command to prevent further distribution traffic over the conversation. After any waiting CMUs have been exchanged, the conversation will be deallocated.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines or procedures:
  - from "DS_SEND_SENDING" on page 156
    - QUERY_FLOW_CONTROL
  - from the Operator
    - OPERATOR_QUIESCE_CONVERSATION
  - from "DS_SEND_CLEANUP_DMU" on page 166
    - DMU_SENT
  - from "DS_SEND_SEND_DMU_NO_MU_ID" on page 181
    - DMU_SENT
  - from "DS_SEND_RECEIVING" on page 190
    - RECEIVING
    - CRMU(TERM_CONVERSATION bit)

	States			
	RESET	CMU	TERM CONV	QUIESCE
Inputs	01	02	03	04
QUERY FLOW CONTROL	-a	-b	-c	-b
RECEIVING	-	1	-	-
OPERATOR QUIESCE CONVERSATION	4	4	4	-
CRMU(TERM CONV=YES)	3	3	-	-
CRMU(TERM CONV=NO)	-	1	1	-
DMU SENT	2	/	-	-

Output Code	Function
a	Signal caller with SEND_DISTRIBUTION.
b	Signal caller with SEND_CONTROL_MU.
c	Signal caller with TERMINATE_CONVERSATION.

DS_SEND_MU_ID_REGISTRY

This FSM manages DS_Send's MU_ID registry for a single connection. Its inputs and actions are listed below. All inputs except INSPECT and INSTANCE_NUMBER return either MU_ID_OP_OK or MU_ID_OP_NOT_OK.

- Assign

  Allocates an entry in the MU_ID registry and assigns to it the next available *MU_ID* value, enters the distribution location information, sets the *MU_ID state* to IN_TRANSIT and sets the *instance number* to ONE (1). The *MU_ID* field in the appropriate next-DSU queue entry is also updated.

- Suspend

  Sets the *MU_ID state* to be SUSPENDED.

- CQMU_Pending

  Sets the *MU_ID state* to be CQMU_PENDING.

- In_Transit

  Sets the *MU_ID state* to be IN_TRANSIT and increments the *instance number*.

- Inspect

  Returns the designated *MU_ID state* (NOT_ASSIGNED, IN_TRANSIT, TRANSFER_PENDING, CQMU_PENDING, RETRY_PENDING, TERMINATION_PENDING, SUSPENDED or PURGED) or, if the MU_ID registry is inaccessible, MU_ID_OP_NOT_OK. *MU_ID*s less than the smallest *MU_ID* contained in the registry return PURGED. *MU_ID*s greater than the highest *MU_ID* contained in the registry return NOT_ASSIGNED.

- Purge

  Sets the *MU_ID state* to be PURGED and removes the *MU_ID* entry's distribution location information. Also, the *MU_ID* field in the appropriate next-DSU queue entry is set to null. Note that setting an *MU_ID state* to PURGED signifies only the breaking of the *MU_ID*-to-distribution association; nothing is implied about the fate of the distribution. The distribution may be retried with a different *MU_ID*, or the operator may reroute it, or the DSU may terminate it and issue a distribution report (if appropriate), or the DSU may simply discard it because the partner has accepted responsibility.

  Entries containing the *MU_ID state* PURGED may be aged out of the registry.

- Retry_Pending

  Sets the *MU_ID state* to be RETRY_PENDING.

- Termination_Pending

  Sets the *MU_ID state* to be TERMINATION_PENDING.

- Transfer_Pending

  Sets the *MU_ID state* to be TRANSFER_PENDING.

- Instance_Number

  Compares the caller's and MU_ID registry's *instance numbers*. MU_ID registry failures preventing the comparison result in MU_ID_OP_NOT_OK being returned to the caller. If the caller's value is less than that in the registry, INSTANCE_NUM_NOT_OK is returned. If the *MU_ID* owning the *instance number* is not in the registry (it is not yet in the registry because its value is too high, or it was once in the registry but has been aged out), INSTANCE_NUM_OK is returned. Otherwise, INSTANCE_NUM_OK is returned.

UPM_CHECK_DUP_CONV_FAIL_REPORT

This procedure is not explicitly specified. As discussed elsewhere (see the discussion for DS_SEND_CHECK_CONV_FAIL, under "Program Structure" on page 146), some conversation failures cause both DS_Send and DS_Receive to initiate exception processing for the same *MU_ID*. This procedure is signalled by DS_SEND_CHECK_CONV_FAIL to determine whether a given REMU represents such a duplicate (and extraneous) exception report for an already-known conversation failure.

If the just-received REMU is reporting a conversation failure, and *MU_ID state* is TERMINATION_PENDING or RETRY_PENDING, and DS_Send's last action for this *MU_ID* was to send a SEMU reporting conversation failure, then DUP_CONV_FAIL_REPORT is returned to the caller. Otherwise, NOT_DUP_CONV_FAIL_REPORT is returned.

DS Receive FSMs

Data Structures

DS_RCV uses the following data structures:

MU_ID Registry: The MU_ID registry is a safe-stored data structure that records *MU_ID*, *MU_ID state* and MU_ID-to-next_DSU_queue-entry mappings. It may be viewed as an array, each element of which contains four entries. The first entry is an *MU_ID*. The second is the *state* of the *MU_ID*. The third is a *distribution locator*, which is used to locate the distribution in the Mid-MU Restart queue. The fourth is the *MU_ID's instance number*.

When a DTMU *prefix* is received, the *MU_ID* is entered in the MU_ID registry, and the entry written to safe store. As the MU changes states, this entry is updated. When the distribution is purged, the distribution locator is replaced by a null entry.

MU_IDs are assigned sequentially by the partner. Only contiguous blocks of entries containing the registry's lowest *MU_IDs* that are in the PURGED state and have null distribution locators are purged from the registry automatically (and their space reclaimed). All other entries are retained, unless explicitly purged by an operator.

Program Structure

A logical decomposition of DS_Receive is given in figure Figure 46 on page 231. The following classes of routines exist in DS_Receive:

- DS_Receive Manager

  The only FSM in this class of routines is "DS_RCV_MANAGER" on page 232. It manages the transition between the LU 6.2 send and receive states, and also checks for the conversation idle condition, which causes the conversation to be terminated.

- Receive-State Manager

  The only FSM in this class of routines is "DS_RCV_RECEIVING" on page 237. This routine manages DS_Receive while in the LU 6.2 receive state. It repeatedly receives DTMUs, DRMUs, DCMUs, SEMUs, CQMUs, and PRMUs from the partner DSU, parses them, and signals the appropriate handler. This routine returns to the caller only when the conversation terminates (normally or abnormally) or DS_Receive enters the LU 6.2 send state.

- Send-State Manager

  The only FSM in this class of routines is "DS_RCV_SENDING" on page 234. This routine is active whenever DS_Receive is in the LU 6.2 send state. It repeatedly reads CRMUs and REMUs from the control MU queue, and sends them to the partner DSU. When the control MU queue is empty, it returns control to the partner.

- Distribution Receiving

  Different actions must be taken depending on the distribution's high- or basic-integrity classification. These FSMs determine the distributions integrity (by checking for the presence or absence of the *MU_ID* value) and trigger the appropriate actions. The FSMs in this class are:

  - "DS_RCV_RECEIVE_DMU" on page 240.

    This FSM implements some high-integrity actions as well as some distribution receiving actions. The distribution receiving action performed by this FSM is simply to signal "DS_RCV_RECEIVE_DMU_NO_MU_ID" on page 259. if the distribution uses basic integrity.

  - "DS_RCV_MU_ID_HANDLER" on page 244.

    This FSM implements some high-integrity actions as well as some distribution receiving actions. The distribution receiving action performed by this FSM is simply to return a signal indicating that the distribution uses only basic integrity. Basic integrity is indicated by the lack of an *MU_ID* in the distribution's encoding.

- CQMU Receiving

  The only FSM in this class of routines is "DS_RCV_CQMU_HANDLER" on page 265. This routine generates a CRMU containing the *MU_ID state* of the *MU_ID* contained in the just-received CQMU. *MU_IDs* higher than the registry's highest received *MU_ID* are considered to be NOT_RECEIVED. *MU_IDs* already purged from the registry are considered to be PURGED.

- SEMU Receiving

  The only FSM in this class of routines is "DS_RCV_SEMU_HANDLER" on page 269. This routine first checks the SEMU's *instance number*; if the *instance number* is too low, the SEMU is discarded and no further actions are taken. If the *instance number* is acceptable, the actions taken in response to a SEMU depend on the *MU_ID state* (e.g., NOT_RECEIVED, TERMINATED, or SUSPENDED) and the retriability or non-retriability of the SEMU's *report code.* Upon return from this FSM, the distribution will be either suspended or terminated. If suspended, the distribution is waiting to be restarted via a DCMU, the *MU_ID state* is SUSPENDED and a CRMU(SUSPENDED) has been generated. If terminated, the distribution has been deleted from the mid-MU restart queue, the server object has been deleted, the *MU_ID state* is TERMINATED and a CRMU(TERMINATED) has been generated.

- PRMU Receiving

  The only FSM in this class of routines is "DS_RCV_PRMU_HANDLER" on page 272. This routine examines DS_Receive's *MU_ID state* of the PRMU's *MU_ID*. If DS_Receive's *MU_ID state* is COMPLETED, TERMINATED or SUSPENDED, DS_Receive changes its *MU_ID state* to PURGED. Otherwise, the PRMU is ignored.

- Bad/Unknown MU Receiving

  The only FSM in this class of routines is "DS_RCV_SEND_ERR_REMU" on page 248. (This FSM is also used for some high-integrity exception actions, which are discussed below.) This FSM is called when a just-received MU does not fall into any of the known MU categories (DTMU, DRMU, DCMU, SEMU, CQMU, or PRMU). The exception action is to issue an LU 6.2 Send_Error, generate and send a REMU reporting the exception (and any other CMUs awaiting transmission), and deallocate the conversation.

- Basic-Integrity Actions

  The only FSM in this class of routines is "DS_RCV_RECEIVE_DMU_NO_MU_ID" on page 259. This routine is similar to DS_RCV_RECEIVE_DMU, except that all *MU_ID* and mid-MU restart processing is omitted, any exceptions cause the distribution to be discarded and no CMUs are used to report the success or failure of the distribution transfer.

- High-Integrity Actions

  These FSMs receive high-integrity distributions from the partner DSU. This involves

  - Checking the distribution's *MU_ID* and *instance number.*
  - Entering the *MU_ID* into the MU_ID registry (if appropriate).
  - Setting the *MU_ID state* to IN_TRANSIT.
  - Receiving and processing the entire distribution.
  - Signalling the responsibility acceptance FSMs.

  All exceptions are handled by the appropriate exception-handling routine, and the results of the reception are returned to the caller. The FSMs in this class are:

  - "DS_RCV_RECEIVE_DMU" on page 240.

    This FSM, assuming the *MU_ID* can be honored correctly, repeatedly receives, parses, and processes the distribution. An entry is made on the mid-MU restart queue and manipulated as appropriate. The server object is written to the server. After the *suffix* has been received, the responsibility acceptance actions are performed to take responsibility for the distribution. Exceptions are processed by the appropriate exception handler, and the success or failure of the transmission is returned to the caller.

  - "DS_RCV_MU_ID_HANDLER" on page 244.

    This routine is signalled by DS_RCV_RECEIVE_DMU to process the distribution's *MU_ID* and *instance number*. If the MU is a DTMU or a DRMU, the *MU_ID state* must be NOT_RECEIVED and the *instance number* must be ONE (1). If the MU is a DCMU, the *MU_ID state* must be SUSPENDED, and the *instance number* must be greater than the previously-received *instance number*. If the *MU_ID* and *instance number* are acceptable, the *MU_ID state* is changed to IN_TRANSIT. The success or failure of the *MU_ID* operations are returned to the caller. If the *MU_ID* is invalid, DS_Receive will inform its partner DSU of the exception and deallocate the conversation. Low *instance numbers* will cause the caller to ignore the distribution. Failures of the MU_ID registry will cause DS_Receive to deallocate the conversation immediately.

- High-Integrity Exception Actions

  These FSMs take the appropriate recovery actions for any exception that might occur while a distribution is being received from the partner DSU. The FSMs in this class are:

  - "DS_RCV_SEND_ERR_REMU" on page 248.

    This FSM issues an LU 6.2 Send_Error verb to interrupt the partner's transmission and generates a REMU describing an exception condition detected by DS_Receive. It is used for "unrecognized MU type," "MU_ID terminated," and "MU_ID state mismatch exceptions," since the exception actions for these cases are identical. "Bad," or unrecognized MUs are discussed above under "Bad/Unknown MU Receiving." An *MU_ID state* mismatch exists for a DTMU or DRMU if the *MU_ID state* is SUSPENDED. An *MU_ID state* mismatch exists for a DCMU if the *MU_ID state* is NOT_RECEIVED. An "MU_ID terminated" condition exists for a DTMU, DRMU or DCMU if the *MU_ID state* is TERMINATED. The exception actions taken by this FSM are:

    - To stop the partner's transmission by using a Send_Error LU 6.2 verb.
    - To generate a REMU informing the partner of the exception.

    In addition to these actions, "MU_ID state mismatch" condition causes DS_Receive to send all waiting CMUs to the partner, and deallocate the conversation.

  - "DS_RCV_SEND_ERR_CRMU" on page 256.

    This FSM issues an LU 6.2 Send_Error verb to interrupt the partner's transmission and generates a CRMU informing the partner that the

MU_ID just received is in either the COMPLETED or PURGED state. The exception actions taken by this FSM are:

- To stop the partner's transmission by using a Send_Error LU 6.2 verb.
- To generate a CRMU informing the partner of the *MU_ID state.*

- "DS_RCV_SEND_ERR" on page 246.

  This FSM issues an LU 6.2 Send_Error verb to interrupt the partner's transmission. When using parallel sessions, network delays may cause MUs to arrive at the receiving DSU in an order different from the order sent by the partner DSU. When such MUs are associated with different *MU_ID*s, the reordering is inconsequential. When reordered MUs refer to the same *MU_ID*, the reordering may affect the processing of the *MU_ID.* If the reordered MUs are only CMUs, the protocols are robust enough to recover and allow the *MU_ID*s to be processed properly (possibly at the cost of extra CMUs being generated). If the reordering involves distribution MUs, the situation is more serious. For example, suppose a tardy REMU were associated with the wrong DCMU: the DS_Send process might initiate exception processing and attempt to terminate the distribution while DS_Receive was forwarding it on to the destination. To handle this latter case, each DMU is assigned an *instance number*, and CMUs carry the *instance number* of the DMU to which they refer. CMUs with obsolete *instance number*s are simply ignored.

  DMUs with obsolete *instance number*s are ignored logically. An LU 6.2 Send_Error verb is issued to terminate the transmission (to save bandwidth), but no REMU is generated. Of course, DS_Receive enters the LU 6.2 send state upon issuing the Send_Error, and thus gets the opportunity to send any waiting CMUs.

- "DS_RCV_SUSP_TERM" on page 250.

  This FSM suspends or terminates (as appropriate) a distribution. When a DS_Send is transmitting a DMU and encounters an exception condition, it issues an LU 6.2 Send_Error verb to inform DS_Receive that the DMU's transmission is being interrupted. Upon receiving this Send_Error, this FSM is signalled to attempt suspending the distribution for later restart via a DCMU. If this suspension is successful, the *MU_ID state* is set to SUSPENDED. If such a suspension is not possible, DS_Receive deletes the distribution and puts the *MU_ID* into TERMINATED state.

  This FSM is also signalled if, while receiving a distribution, DS_Receive detects that the partner DSU has violated DS's use of the LU 6.2 basic conversation verbs. An example of such a protocol error is for DS_Send to issue a Receive_And_Wait verb while transmitting a distribution. The distribution being received is discarded; the *MU_ID* is put into TERMINATED state.

- "DS_RCV_SEND_ERR_SUSP_TERM_REMU" on page 252.

  This FSM issues an LU 6.2 Send_Error verb to interrupt the partner's transmission, suspends or terminates the distribution (as appropriate), and generates a REMU describing the exception condition detected by

DS_Receive. It is signalled if DS_Receive, while receiving a distribution, detects an exception in the queue manager, parser, or server. It is also signalled if DS_Receive, after successfully receiving a distribution, fails to accept responsibility for it.

An LU 6.2 Send_Error is issued to terminate the distribution's transmission. If the exception is restartable via a DCMU, and DS_Receive is capable of restarting the distribution, the distribution is suspended on the mid-MU restart queue, the server object is saved, and the *MU_ID* is put into SUSPENDED state. Otherwise, the distribution is discarded and the *MU_ID* is put into TERMINATED state. A REMU is generated to inform the partner DSU of the exception.

– "DS_RCV_REMU_SUSP_TERM" on page 254.

  This FSM generates a REMU describing an exception condition detected by DS_Receive, and suspends the distribution (to be restarted via a DCMU) or discards it (if suspension is not possible or appropriate). It is signalled whenever DS_Receive is receiving a distribution and the LU 6.2 conversation with the partner fails. Such failures can be detected not only from the Receive_And_Wait verbs issued to receive the DMU, but from Send_Error verb issued during DS_RCV_SEND_ERR_SUSP_TERM_REMU processing.

  If the distribution can be restarted via a DCMU, it and the associated server object are saved and the *MU_ID* put into SUSPENDED state. Otherwise, the distribution is discarded and the *MU_ID* put into TERMINATED state.

- Responsibility Acceptance Actions

  The only FSM in this class of routines is "DS_RCV_ENQ_SCHED" on page 262. This FSM is signalled to accept responsibility of a distribution (either basic or high integrity) that has been received successfully. It creates a router-director queue entry for the distribution, schedules DS_Router_Director, changes the *MU_ID state* to COMPLETED and releases the just-created router-director queue entry. If all these operations succeed, a CRMU(COMPLETED) is generated. If any of the above operations fail, the distribution is left unchanged, just as it was when the FSM was signalled--no entry is left on the router-director queue, and the *MU_ID state* is left in IN_TRANSIT. (And any operations that were attempted are backed out.)

- DS_Receive Utility Routines

  These FSMs perform relatively simple, well-defined functions that are needed by various other FSMs in DS_Receive. The FSMs in this class are:

  – "DS_RCV_SUSP_DIST" on page 274.

    This FSM suspends a partially received distribution. That is, the distribution's entry in the mid-MU restart queue is released (so that it may be accessed later, probably by a different instance of DS_Receive), and a Terminate_Write server verb is issued (so that the server object may be accessed later).

- "DS_RCV_DISCARD_DIST" on page 276.

  This FSM discards a distribution. The server object is deleted (and restartability terminated), and the distribution's mid-MU restart queue entry is dequeued and discarded.

- "DS_RCV_SEND_CONVERSATION_MGR" on page 278.

  This FSM sends a single buffer to the partner DSU by issuing an LU 6.2 Send_Data verb. The outcome of the Send_Data (LU 6.2 accepted the data without detecting an exception, a conversation failure was detected, the partner DSU issued a Send_Error, etc.) is returned to the caller.

- "DS_RCV_MU_ID_REGISTRY" on page 280.

  This procedure manages DS_Receive's access to the MU_ID registry. An *MU_ID state* may be assigned or inspected, or a just-received *instance number* may be compared to the *instance number* in the registry. (An MU containing an *instance number* that is too low is ignored.)

- "PREPARSER" on page 280

  This procedure inspects the initial LLID of an MU and returns the MU type (DTMU, DRMU, DCMU, SEMU, PRMU, or CQMU). It also identifies and returns the *MU_ID* and *instance number*, if appropriate. CMUs are parsed completely; if the CMU contains a format exception, or if an *MU_ID* or *instance number* is missing when required or otherwise invalid, BAD_MU is returned. If the initial LL's ID is not recognized, UNKNOWN_MU is returned.

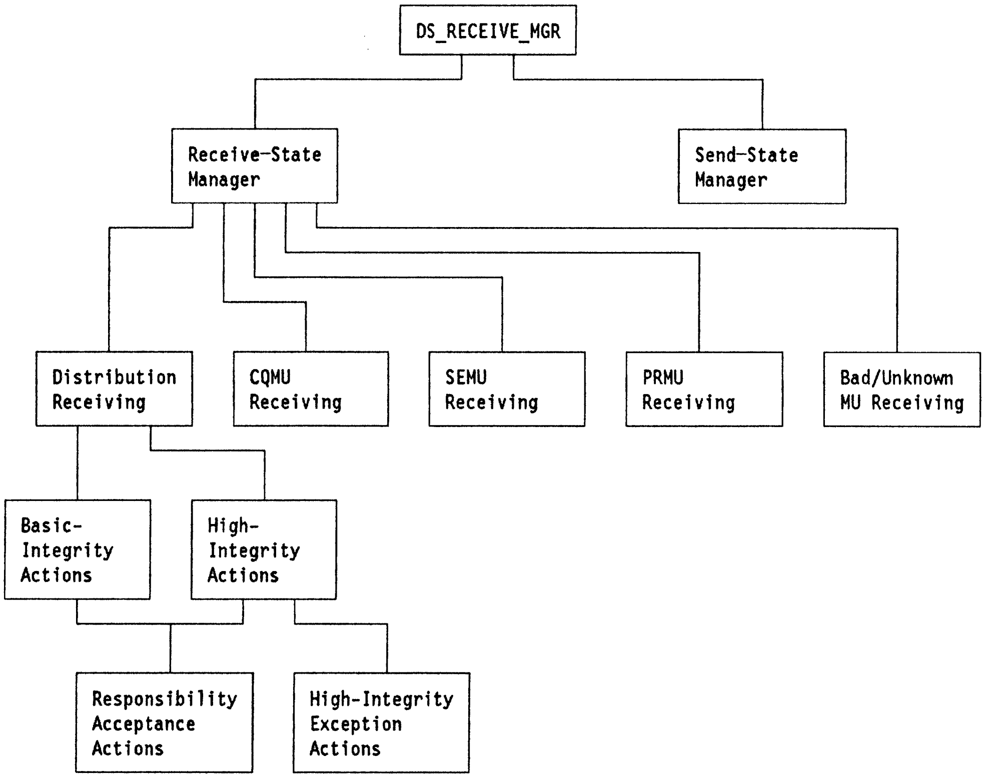


Figure 46. DS_Receive Logical Structure

DS_RCV_MANAGER

Function: This finite-state machine describes the functional processing for DS_Receive. This FSM is started by LU 6.2 on receiving an Attach, or by the operator via a START_TRANSACTION. If DS_Receive is started by LU 6.2 via Attach, it is initially in the LU 6.2 receive state. Otherwise, it is initially in the LU 6.2 send state.

The primary processing of this FSM is done in states 3 (RECEIVING), 4 (SENDING) and 5 (IDLE TEST). When in RECEIVING state (state 3), DS_Receive is in the LU 6.2 receive state and is receiving distribution and control MUs from the partner. If DS_Receive then receives the change direction indication from LU 6.2, it immediately attempts to send any waiting control MUs to the partner by signalling DS_RCV_SENDING.

When in SENDING state (state 4), DS_Receive is in the LU 6.2 send state, and is sending CMUs to the partner. The lower-level FSMs signal DS_Receive to enter the LU 6.2 receive state by returning CHANGE_DIRECTION. DS_RCV_MANAGER first checks to determine if an idle conversation exists by signalling IDLE_DETECTOR. If an idle conversation does exist, the conversation is simply deallocated. If the conversation is not idle, DS_Receive goes into the LU 6.2 receive state by signalling DS_RCV_RECEIVING.

An idle conversation exists whenever both of the following conditions hold:

- When DS_Receive was last in the LU 6.2 receive state, it received no MUs from the partner DS_Send.
- DS_Receive, which is currently in the LU 6.2 send state, has had no control MUs to send to the partner DS_Send.

This FSM gets control from one of the following:

- Signals from higher-level finite-state machines or procedures:
  - from "FSM_SCHED_MGR" on page 351
    - START_TRANSACTION
- Signals from lower-level DS_Receive finite-state machines:
  - from "DS_RCV_SENDING" on page 234
    - CHANGE_DIRECTION
    - DEALLOCATE_LOCAL
    - DEALLOCATE_FLUSH
    - DEALLOCATE_ABEND
  - from "DS_RCV_RECEIVING" on page 237
    - CHANGE_DIRECTION
    - DEALLOCATE_LOCAL
    - DEALLOCATE_ABEND
    - SEND_CMU_AND_DEALLOCATE
- Signals from machines providing common services:
  - from "IDLE_DETECTOR" on page 284
    - CHANGE_DIRECTION
    - DEALLOCATE_FLUSH
- Signals from LU 6.2 presentation services
  - OK
  - ATTACH
  - ALLOCATION_ERROR
  - ALLOC_PARM_ERROR

	States						
	RESET	ALLOC	REC'VING	SENDING	IDLE TEST	SEND CMU	DEALL PEND
Inputs	01	02	03	04	05	06	07
START TRANSACTION	2a	/	/	/	/	/	/
ATTACH	3b	/	/	/	/	/	/
OK	/	4c	/	/	/	/	1
ALLOCATION ERROR	/	7d	/	/	/	/	/
ALLOC PARM ERROR	/	1e	/	/	/	/	/
CHANGE DIRECTION	/	/	4c	5h	3j	7i	/
DEALLOCATE LOCAL	/	/	7f	7f	/	7f	/
DEALLOCATE FLUSH	/	/	/	7i	7i	7i	/
DEALLOCATE ABEND	/	/	7g	7g	/	7f	/
SEND CMU AND DEALLOCATE	/	/	6c	/	/	/	/

Output Code	Function
a	Signal LU 6.2 presentation services with ALLOCATE to establish the conversation with DS_Send.
b	Signal LU 6.2 presentation services with GET_ATTRIBUTES. Signal DS_RCV_RECEIVING with START.
c	Signal DS_RCV_SENDING with SEND.
d	Notify operations of the exception condition. Signal LU 6.2 presentation services with DEALLOCATE specifying *type*(LOCAL).
e	Notify operations of the exception condition.
f	Signal LU 6.2 presentation services with DEALLOCATE specifying *type*(LOCAL).
g	Signal LU 6.2 presentation services with DEALLOCATE specifying *type*(ABEND).
h	Signal IDLE_DETECTOR with CHANGE_DIRECTION.
i	Signal LU 6.2 presentation services with DEALLOCATE specifying *type*(FLUSH).
j	Signal DS_RCV_RECEIVING with START.

DS_RCV_SENDING

Function: This finite-state machine describes the functional processing for DS_Receive while it is in the LU 6.2 send state. This FSM repeatedly finds a CMU on the control MU queue, sends it, and discards the queue entry. FSM IDLE_DETECTOR is also signalled. When the queue is emptied, a CHANGE_DIRECTION signal is returned to the caller.

If the LU 6.2 conversation failure is detected, the CMU being sent is left on the queue to be sent by another instance of DS_Receive.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines or procedures:
  - from "DS_RCV_MANAGER" on page 232
    - SEND
- Signals from machines providing common services:
  - from "DS_RCV_SEND_CONVERSATION_MGR" on page 278
    - OK
    - CONVERSATION_FAILURE
    - PROG_ERROR
    - PROTOCOL_ERROR
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK
    - QUEUE_EMPTY
  - from "IDLE_DETECTOR" on page 284
    - nothing returned from this FSM

	States					
	RESET	READQ	SEND CMU	DISC CMU	CONV FAIL	PROTO ERROR
Inputs	01	02	03	04	05	06
SEND	2a	/	/	/	/	/
QUEUE OK	/	3b	/	2a	1g	1i
QUEUE NOT OK	/	1c	/	1c	1h	1j
QUEUE EMPTY	/	1d	/	/	/	/
OK	/	/	4e	/	/	/
CONVERSATION FAILURE	/	/	5f	/	/	/
PROG ERROR	/	/	6f	/	/	/
PROTOCOL ERROR	/	/	6f	/	/	/

Output Code	Function
a	Signal QUEUE_MGR with READQ, specifying *queue*(CONTROL_MU_QUEUE).
b	Signal IDLE_DETECTOR with SOMETHING_SENT. Signal DS_RCV_SEND_CONVERSATION_MGR with SEND_BUFFER to send the control MU to the partner DSU.
c	Notify operations of the exception condition. Signal QUEUE_MGR with HOLD specifying *queue*(CONTROL_MU_QUEUE) for this connection. Nothing is to be sent to the partner DSU. Signal caller with DEALLOCATE_FLUSH.
d	Signal caller with CHANGE_DIRECTION.
e	Signal QUEUE_MGR with DEQ to discard the control MU.
f	Signal QUEUE_MGR with RELEASEQ, freeing the control MU for resending later.
g	Signal caller with DEALLOCATE_LOCAL.
h	Notify operations of the exception condition. Signal QUEUE_MGR with HOLD specifying *queue*(CONTROL_MU_QUEUE) for this connection. Nothing is to be sent to the partner DSU. Signal caller with DEALLOCATE_LOCAL.
i	Notify operations of the exception condition. Signal caller with DEALLOCATE_ABEND.
j	Notify operations of the exception condition. Signal QUEUE_MGR with HOLD specifying *queue*(CONTROL_MU_QUEUE) for this connection. Nothing is to be sent to the partner DSU. Signal caller with DEALLOCATE_ABEND.

DS_RCV_RECEIVING

Function: This finite-state machine controls DS_Receive in the LU 6.2 receive state. A buffer is received from LU 6.2, and the initial LLID examined (states 1 and 2). This LLID indicates that the MU being received is a DTMU, DCMU, DRMU, CQMU, PRMU, SEMU, or an unknown MU type. An appropriate handler is called for each MU type (state 3). The handler's results are returned to the caller. If an exception occurs (such as an LU 6.2 conversation failure), it is also returned to the caller.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines or procedures:
  - from "DS_RCV_MANAGER" on page 232
- Signals from lower-level DS_Receive finite-state machines:
  - from "DS_RCV_RECEIVE_DMU" on page 240
    - MU_OK
    - CONVERSATION_FAILURE
    - SEND_SIDE_EXCEPT
    - RCV_SIDE_EXCEPT
    - PROTOCOL_ERROR
    - MU_ID_STATE_MISMATCH
    - DS_RCV_SYSTEM_ERROR
  - from "PREPARSER" on page 280
    - DMU
    - CQMU
    - PRMU
    - SEMU
    - UNKNOWN_MU
    - BAD_CMU
  - from "DS_RCV_CQMU_HANDLER" on page 265
    - MU_OK
    - MU_NOT_OK
  - from "DS_RCV_PRMU_HANDLER" on page 272
    - MU_OK
    - MU_NOT_OK
  - from "DS_RCV_SEMU_HANDLER" on page 269
    - MU_OK
    - MU_NOT_OK
  - from "DS_RCV_SEND_ERR_REMU" on page 248
    - CONVERSATION_FAILURE
    - UNREC_MU_TYPE
    - DS_RCV_SYSTEM_ERROR
- Signals from machines providing common services:
  - from "RCV_BUFFER_MGR" on page 282
    - OK
    - CHANGE_DIRECTION
    - CONVERSATION_FAILURE
    - PROG_ERROR
    - PROTOCOL_ERROR
    - DEALLOCATE_NORMAL
  - from "IDLE_DETECTOR" on page 284
    - no signals received from this FSM

	States							
	RESET	RCV BUFF	MU TYPE	DMU	CQMU	PRMU	SEMU	BAD MU MU_ID
Inputs	01	02	03	04	05	06	07	08
START	2a	/	/	/	/	/	/	/
OK	/	3b	/	/	/	/	/	/
CHANGE DIRECTION	/	1c	/	/	/	/	/	/
PROG ERROR	/	-a	/	/	/	/	/	/
DEALLOCATE NORMAL	/	1d	/	/	/	/	/	/
CONVERSATION FAILURE	/	1d	/	1d	/	/	/	1d
PROTOCOL ERROR	/	1e	/	1e	/	/	/	/
SEND SIDE EXCEPT	/	/	/	2a	/	/	/	/
RCV SIDE EXCEPT	/	/	/	1c	/	/	/	/
DS RCV SYSTEM ERROR	/	/	/	1e	/	/	/	1e
MU ID STATE MISMATCH	/	/	/	1k	/	/	/	/
UNREC MU TYPE	/	/	/	/	/	/	/	1k
DMU	/	/	4f	/	/	/	/	/
CQMU	/	/	5g	/	/	/	/	/
PRMU	/	/	6h	/	/	/	/	/
SEMU	/	/	7i	/	/	/	/	/
UNKNOWN MU	/	/	8j	/	/	/	/	/
BAD CMU	/	/	1e	/	/	/	/	/
MU OK	/	/	/	2a	2a	2a	2a	/
MU NOT OK	/	/	/	/	1e	1e	1e	/

Output Code	Function
a	Signal RCV_BUFFER_MGR with RECEIVE_BUFFER.
b	Signal IDLE_DETECTOR with SOMETHING_RECEIVED. Signal PREPARSER with RETURN_MU_TYPE. (For control MUs, the PREPARSER either parses the CMU completely, or signals the parser to complete the parsing.)
c	Signal caller with CHANGE_DIRECTION.
d	Signal caller with DEALLOCATE_LOCAL.
e	Notify operations of the exception condition. Signal caller with DEALLOCATE_ABEND.
f	Signal DS_RCV_RECEIVE_DMU with DECODE.
g	Signal DS_RCV_CQMU_HANDLER with START.
h	Signal DS_RCV_PRMU_HANDLER with START.
i	Signal DS_RCV_SEMU_HANDLER with START.
j	Signal DS_RCV_SEND_ERR_REMU with UNREC_MU_TYPE.
k	Notify operations of the exception condition. Signal caller with SEND_CMU_AND_DEALLOCATE.

DS_RCV_RECEIVE_DMU

Function: This is DS_Receive's "distribution handler." It receives and parses a high-integrity DTMU, DRMU or DCMU, or signals DS_RCV_RECEIVE_DMU_NO_MU_ID to handle a basic-integrity DTMU or DRMU. The major processing in this FSM is in two loops: states 4 and 5 are a loop for receiving and parsing the DMU. States 4, 5 and 6 are a loop for receiving and writing the server object. When the DMU's *suffix* has been received successfully, a Terminate_Write server verb is issued and DS_RCV_ENQ_SCHED is signalled to accept responsibility for the distribution.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines:
  - from "DS_RCV_RECEIVING" on page 237
    - DECODE
- Signals from lower-level DS_Receive finite-state machines:
  - from "DS_RCV_ENQ_SCHED" on page 262
    - ENQ_SCHED_OK
    - ENQ_SCHED_NOT_OK
    - ENQ_SCHED_OK_DEALLOCATE
  - from "DS_RCV_MU_ID_HANDLER" on page 244
    - MU_ID_OK
    - BAD_INSTANCE_NUM
    - MU_ID_NOT_USED
    - DS_RCV_SYSTEM_ERROR
    - MU_ID_STATE_MISMATCH
    - MU_ID_TERMINATED
    - MU_ID_COMP_PURG
  - from "DS_RCV_SEND_ERR_REMU" on page 248.
    - CONVERSATION_FAILURE
    - MU_ID_STATE_MISMATCH
    - DS_RCV_SYSTEM_ERROR
    - MU_ID_TERMINATED
  - from "DS_RCV_SEND_ERR_CRMU" on page 256.
    - CONVERSATION_FAILURE
    - MU_ID_COMP_PURG
    - DS_RCV_SYSTEM_ERROR
  - from "DS_RCV_RECEIVE_DMU_NO_MU_ID" on page 259.
    - MU_OK
    - DS_RCV_SYSTEM_ERROR
    - RCV_SIDE_EXCEPT
    - SEND_SIDE_EXCEPT
    - CONVERSATION_FAILURE
    - PROTOCOL_ERROR
  - from "DS_RCV_SEND_ERR" on page 246.
    - SEND_ERROR_GENERATED
    - CONVERSATION_FAILURE
  - from "DS_RCV_SEND_ERR_SUSP_TERM_REMU" on page 252.
    - DS_RCV_SYSTEM_ERROR
    - SEND_ERR_SUSP_TERM_REMU
    - CONVERSATION_FAILURE

- from “DS_RCV_SUSP_TERM” on page 250.
    - PROG_ERR_RCVD
    - DS_RCV_SYSTEM_ERROR
    - PROTOCOL_ERROR_RCVD
- from “DS_RCV_REMU_SUSP_TERM” on page 254.
    - DS_RCV_SYSTEM_ERROR
    - REMU_SUSP_TERM

- Signals from machines providing common services:
    - from “RCV_BUFFER_MGR” on page 282
        - OK
        - CONVERSATION_FAILURE
        - PROTOCOL_ERROR
        - DEALLOCATE_NORMAL
        - PROG_ERROR
        - CHANGE_DIRECTION
    - from “QUEUE_MGR” on page 357
        - QUEUE_OK
        - QUEUE_NOT_OK
    - “SERVER_MGR” on page 354
        - OBJECT_OK
        - OBJECT_NOT_OK
        - SPECIFIC_SERVER_EXCEPTION
- Signals from “PARSER” on page 359:
    - PARSE_OK
    - PARSE_OK_OBJECT
    - PARSE_COMPLETE
    - PARSE_NOT_OK

	States									
	RESET	MID MU REST	CHECK MU_ID	PARS	RCV NEXT	OBJ PEND	END DMU	ACC PEND	PROC ERR	NO MU_ID
Inputs	01	02	03	04	05	06	07	08	09	10
DECODE	2a	/	/	/	/	/	/	/	/	/
MU ID OK	/	3b	/	/	/	/	/	/	/	/
MU ID TERMINATED	/	9c	/	/	/	/	/	/	1s	/
MU ID STATE MISMATCH	/	9v	/	/	/	/	/	/	1w	/
MU ID COMP PURG	/	9x	/	/	/	/	/	/	1s	/
MU ID NOT USED	/	10d	/	/	/	/	/	/	/	/
BAD INSTANCE NUM	/	9e	/	/	/	/	/	/	/	/
QUEUE OK	/	/	4g	/	/	/	/	/	/	/
QUEUE NOT OK	/	/	9h	/	/	/	/	/	/	/
PARSE OK	/	/	/	5i	/	/	/	/	/	/
PARSE OK OBJECT	/	/	/	6j	/	/	/	/	/	/
PARSE COMPLETE	/	/	/	7k	/	/	/	/	/	/
PARSE NOT OK	/	/	/	9h	/	/	/	/	/	/
OK	/	/	/	/	4g	/	/	/	/	/
PROG ERROR	/	/	/	/	9m	/	/	/	/	/
CONVERSATION FAILURE	/	/	/	/	9n	/	/	/	1r	1r
PROTOCOL ERROR	/	/	/	/	9o	/	/	/	/	1u
DEALLOCATE NORMAL	/	/	/	/	9n	/	/	/	/	/
CHANGE DIRECTION	/	/	/	/	9o	/	/	/	/	/
OBJECT OK	/	/	/	/	/	5i	8p	/	/	/
OBJECT NOT OK	/	/	/	/	/	9h	9h	/	/	/
SPECIFIC SERVER EXCEPTION	/	/	/	/	/	9h	/	/	/	/
ENQ SCHED OK	/	/	/	/	/	/	/	1q	/	/
ENQ SCHED OK DEALLOC	/	/	/	/	/	/	/	1f	/	/
ENQ SCHED NOT OK	/	/	/	/	/	/	/	9h	/	/
REMU SUSP TERM	/	/	/	/	/	/	/	/	1r	1r
SEND ERR SUSP TERM REMU	/	/	/	/	/	/	/	/	1s	1s
RCV SIDE EXCEPT	/	/	/	/	/	/	/	/	1s	1s
PROG ERR RCVD	/	/	/	/	/	/	/	/	1t	1t
SEND SIDE EXCEPT	/	/	/	/	/	/	/	/	1t	1t
PROTOCOL ERROR RCVD	/	/	/	/	/	/	/	/	1u	/
DS RCV SYSTEM ERROR	/	1f	/	/	/	/	/	/	1f	1f
SEND ERROR GENERATED	/	/	/	/	/	/	/	/	1s	/
MU OK	/	/	/	/	/	/	/	/	/	1q

Output Code	Function
a	Signal DS_RCV_MU_ID_HANDLER with the MU type (DTMU, DRMU or DCMU).
b	If the MU is a DTMU and mid-MU restart capability is provided, then signal QUEUE_MGR with WRITEQ, specifying *queue*(MID_MU_RESTART_QUEUE). If this operation fails, implementations may reject the distribution. Or, they may choose to receive the distribution without mid-MU restart capability. The former case is modeled. If the MU is a DCMU, then signal QUEUE_MGR with READQ specifying *queue*(MID_MU_RESTART_QUEUE) to fetch the previously-received portion of the distribution. If the MU is a DRMU, or a DTMU where no mid-MU restart capability is provided, then processing continues as though QUEUE_MGR returned QUEUE_OK.
c	Signal DS_RCV_SEND_ERR_REMU with MU_ID_TERMINATED.
d	Signal DS_RCV_RECEIVE_DMU_NO_MU_ID with START.
e	Signal DS_RCV_SEND_ERR with START.
f	Signal caller with DS_RCV_SYSTEM_ERROR.
g	Signal PARSER with PARSE to parse information received.
h	Signal DS_RCV_SEND_ERR_SUSP_TERM_REMU with START to process the error appropriately.
i	Signal RCV_BUFFER_MGR with RECEIVE_BUFFER to receive information from LU 6.2.
j	Signal SERVER_MGR with WRITE to write the just-received portion of the server object. SERVER_MGR will issue a Initiate_Write, if appropriate.
k	Signal SERVER_MGR with TERMINATE_WRITE with *termination_type* NORMAL to terminate restartability. If no server object has been processed (and therefore no Initiate_Write has been issued), processing continues with OBJECT_OK.
m	Signal DS_RCV_SUSP_TERM with PROG_ERR to process the error appropriately.
n	Signal DS_RCV_REMU_SUSP_TERM with START to process the error appropriately.
o	Signal DS_RCV_SUSP_TERM with PROT_ERR.
p	Signal DS_RCV_ENQ_SCHED with ENQ_SCHED to place the distribution on the router-director queue, schedule DS_Router_Director, set the MU_ID registry entry and build and enqueue the CRMU on the control MU queue.
q	Signal caller with MU_OK to indicate that a DMU has been completely received.
r	Signal caller with CONVERSATION_FAILURE.
s	Signal caller with RCV_SIDE_EXCEPT.
t	Signal caller with SEND_SIDE_EXCEPT to indicate that an error has occurred in DS_Send and that the sender exception protocols should be followed.
u	Signal caller with PROTOCOL_ERROR to indicate that a protocol error has occurred in the conversation between DS_Send and DS_Receive.
v	Signal DS_RCV_SEND_ERR_REMU with MU_ID_STATE_MISMATCH.
w	Signal caller with MU_ID_STATE_MISMATCH to indicate that the received MU (or an *MU_ID_state* implied by the MU) is inconsistent with the *MU_ID_state* in the registry.
x	Signal DS_RCV_SEND_ERR_CRMU with MU_ID_COMP_PURG to indicate that the received *MU_ID* has an *MU_ID state* of COMPLETED or TERMINATED.

DS_RCV_MU_ID_HANDLER

Function: This finite-state machine does the initial *MU_ID* processing when a distribution MU is first received. If no *MU_ID* is present in a DTMU or DRMU, the caller is informed that the distribution does not use *MU_ID*s. If the MU is a DTMU or a DRMU, the *MU_ID state* should be NOT_RECEIVED. Otherwise an error exists.

For DCMUs, the *instance number* is checked; if it is less than or equal to the *instance number* in the MU_ID registry, the DCMU is considered tardy, and ignored. If the *instance number* is greater than the one in the registry, the *MU_ID state* is checked to see that it is SUSPENDED. Otherwise, an error exists.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines or procedures:
  - from "DS_RCV_RECEIVE_DMU" on page 240
    - DTMU
    - DRMU
    - DCMU
- Signals from lower-level DS_Receive finite-state machines:
  - from "DS_RCV_MU_ID_REGISTRY" on page 280
    - NOT_RECEIVED
    - IN_TRANSIT
    - SUSPENDED
    - TERMINATED
    - COMPLETE
    - PURGED
    - INSTANCE_NUM_OK
    - INSTANCE_NUM_EQUAL
    - INSTANCE_NUM_LOW
    - MU_ID_OP_OK
    - MU_ID_OP_NOT_OK
    - MU_ID_NOT_USED
- Signals from machines providing common services:
  - from system operator
    - OP_CONTINUE
    - OP_ABORT

Inputs	States				
	RESET	INST NUM	CONTINUE MU	NEW MU	MU_ID OP
	01	02	03	04	05
DTMU	4a	/	/	/	/
DRMU	4a	/	/	/	/
DCMU	2b	/	/	/	/
INSTANCE NUM HIGH	/	3a	/	/	/
INSTANCE NUM EQUAL	/	1c	/	/	/
INSTANCE NUM LOW	/	1c	/	/	/
NOT RECEIVED	/	/	1e	5f	/
SUSPENDED	/	/	5f	1e	/
TERMINATED	/	/	1i	1i	/
COMPLETE	/	/	1j	1j	/
PURGED	/	/	1j	1j	/
MU ID NOT USED	/	/	/	1g	/
MU ID OP OK	/	/	/	/	1h
MU ID OP NOT OK	/	1d	1d	1d	1d

Output Code	Function
a	Signal DS_RCV_MU_ID_REGISTRY with INSPECT.
b	Signal DS_RCV_MU_ID_REGISTRY with INSTANCE_NUMBER to compare the DCMU's *instance number* with the MU_ID registry's *instance number*. The DCMU's *instance number* must be greater than the MU_ID registry's *instance number*, and the MU_ID registry's number is updated to be equal to the DCMU's.
c	Notify operations of the exception condition. Signal caller with BAD_INSTANCE_NUM.
d	Signal caller with DS_RCV_SYSTEM_ERROR.
e	Notify operations of the exception condition. Signal caller with MU_ID_STATE_MISMATCH.
f	Signal DS_RCV_MU_ID_REGISTRY with IN_TRANSIT.
g	Signal caller with MU_ID_NOT_USED.
h	Signal caller with MU_ID_OK.
i	Notify operations of the exception condition. Signal caller with MU_ID_TERMINATED.
j	Signal caller with MU_ID_COMP_PURG to indicate that the just-received *MU_ID* has an *MU_ID state* of COMPLETED or PURGED.

DS_RCV_SEND_ERR

Function: This finite-state machine is signalled when an exception is encountered while receiving a distribution and the immediate exception action is simply to issue a Send_Error LU 6.2 verb. The Send_Error verb terminates the transmission. (No REMU is generated, the distribution in question is not suspended or discarded, and the *MU_ID state* is not modified.) This FSM is signalled when:

- A DCMU with an obsolete *instance number* is received. Effectively, the DCMU is ignored, but no REMU is generated.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines or procedures:
  - from "DS_RCV_RECEIVE_DMU" on page 240
    - START
- Signals from LU 6.2 presentation services
  - OK
  - DEALLOCATE_NORMAL
  - RESOURCE_FAILURE

	States	
	RESET	**SEND ERR**
Inputs	**01**	**02**
START	2a	/
OK	/	1b
DEALLOCATE NORMAL	/	1c
RESOURCE FAILURE	/	1c

Output Code	Function
a	Signal LU 6.2 presentation services with SEND_ERROR.
b	Signal caller with SEND_ERROR_GENERATED.
c	Signal caller with CONVERSATION_FAILURE.

DS_RCV_SEND_ERR_REMU

Function: This finite-state machine is signalled when an exception has been encountered while receiving a distribution and the immediate exception action is to issue an LU 6.2 Send_Error verb to terminate the transmission, and generate a REMU informing the partner DSU of the exception code. Eventually, DS_Receive will send this REMU (and any other CMUs waiting to be sent) to the partner. The conversation will be deallocated (if appropriate). This FSM is signalled in the following cases:

- by DS_RCV_RECEIVING when an MU of an unrecognized type is received.
- by DS_RCV_RECEIVE_DMU when an MU with an *MU_ID state* mismatch is detected. *MU_ID state* mismatches most often reflect an incompatibility between the DMU and the MU_ID registry. For example, receiving a DTMU whose *MU_ID state* is anything other than NOT_RECEIVED or TERMINATED is such an incompatibility. Receiving a DCMU with a correct *instance number* and an *MU_ID state* of other than SUSPENDED or TERMINATED is another example.
- by DS_RCV_RECEIVE_DMU when an MU_ID-terminated condition is detected. This condition exists whenever a DMU is received, but the *MU_ID state* is TERMINATED. This condition may be the result of a simple race condition, in which a SEMU outruns its associated DMU.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines or procedures:
  - from "DS_RCV_RECEIVING" on page 237
    - UNREC_MU_TYPE
  - from "DS_RCV_RECEIVE_DMU" on page 240
    - MU_ID_STATE_MISMATCH
    - MU_ID_TERMINATED
- Signals from machines providing common services:
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK
- Signals from LU 6.2 presentation services
  - OK
  - ALLOCATION_ERROR
  - RESOURCE_FAILURE

Inputs	States							
	RESET	SEND ERR UNREC	SEND REMU UNREC	SEND ERR MU SEQ	SEND REMU MU SEQ	SEND ERR TERM	SEND REMU ERR TERM	REMU CONV FAIL
	01	02	03	04	05	06	07	08
UNREC MU TYPE	2a	/	/	/	/	/	/	/
MU ID STATE MISMATCH	4a	/	/	/	/	/	/	/
MU ID TERMINATED	6a	/	/	/	/	/	/	/
OK	/	3b	/	5b	/	7b	/	/
DEALLOCATE NORMAL	/	8b	/	8b	/	8b	/	/
RESOURCE FAILURE	/	8b	/	8b	/	8b	/	/
QUEUE OK	/	/	1c	/	1f	/	1g	1e
QUEUE NOT OK	/	/	1d	/	1d	/	1d	1e

Output Code	Function
a	Signal LU 6.2 presentation services with SEND_ERROR.
b	Build a REMU with the appropriate exception code and enqueue it on the control MU queue by signalling QUEUE_MGR with WRITEQ specifying *queue*(CONTROL_MU_QUEUE).
c	Signal caller with UNREC_MU_TYPE.
d	Signal caller with DS_RCV_SYSTEM_ERROR.
e	Signal caller with CONVERSATION_FAILURE.
f	Signal caller with MU_ID_STATE_MISMATCH.
g	Signal caller with MU_ID_TERMINATED.

DS_RCV_SUSP_TERM

Function: This finite-state machine is signalled when an exception has been encountered while receiving a distribution and the immediate exception action is to suspend or discard the distribution (depending on the specific exception and whether the distribution could be restarted with a DCMU). This FSM is signalled when:

- DS_RCV_RECEIVE_DMU expected DMU data from an LU 6.2 Receive_And_Wait verb, but received a PROG_ERROR indication instead. The PROG_ERROR indicates that the partner DSU has encountered some exception condition preventing it from continuing to transmit the DMU. The partner will transmit details of the exception in a SEMU, but until the SEMU is received, DS_Receive interrupts its processing of this distribution.

  DS_Receive presumes that the exception will be retriable, suspending the distribution if possible, and setting the *MU_ID state* to SUSPENDED. This allows the distribution to be restarted via a DCMU if the partner so chooses.

  Otherwise, the distribution is discarded, and the *MU_ID state* set to TERMINATED. In this case, if the exception is retriable, the partner retries it from the beginning with a different *MU_ID*.

- DS_RCV_RECEIVE_DMU detects that the partner DSU has violated DS use of the LU 6.2 basic conversation protocol boundary. Protocol errors are not retriable. The distribution is discarded, and the *MU_ID state* changed to TERMINATED. DS_Receive will deallocate the conversation.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines or procedures:
  - from “DS_RCV_RECEIVE_DMU” on page 240
    - PROG_ERR
    - PROT_ERR
- Signals from lower-level DS_Receive finite-state machines:
  - from “DS_RCV_SUSP_DIST” on page 274
    - DIST_SUSPENDED
    - SUSPEND_FAILED
  - from “DS_RCV_DISCARD_DIST” on page 276
    - DIST_DISCARDED
    - DISCARD_FAILED
  - from “DS_RCV_MU_ID_REGISTRY” on page 280
    - MU_ID_OP_OK
    - MU_ID_OP_NOT_OK
  - from “UPM_EXCEPT_RECOVERY_ACTION” on page 285
    - RETRIABLE_WITH_MID_MU
    - RETRIABLE_WITHOUT_MID_MU

Inputs	States						
	RESET	REC ACT	SUSP DIST	DISC DIST PROG	MU_ID STATE PROG	DISC DIST PROT	MU_ID STATE PROT
	01	02	03	04	05	06	07
PROG ERR	2a	/	/	/	/	/	/
PROT ERR	6c	/	/	/	/	/	/
RETRIABLE WITH MID MU	/	3b	/	/	/	/	/
RETRIABLE WITHOUT MID MU	/	4c	/	/	/	/	/
DIST SUSPENDED	/	/	5d	/	/	/	/
SUSPEND FAILED	/	/	4c	/	/	/	/
DIST DISCARDED	/	/	/	5e	/	7e	/
DISCARD FAILED	/	/	/	5e	/	7e	/
MU ID OP OK	/	/	/	/	1f	/	1h
MU ID OP NOT OK	/	/	/	/	1g	/	1g

Output Code	Function
a	Signal UPM_EXCEPT_RECOVERY_ACTION with the error code.
b	Signal DS_RCV_SUSP_DIST with START.
c	Signal DS_RCV_DISCARD_DIST with START.
d	Signal DS_RCV_MU_ID_REGISTRY with SUSPENDED.
e	Signal DS_RCV_MU_ID_REGISTRY with TERMINATED.
f	Signal caller with SEND_SIDE_EXCEPT.
g	Signal caller with DS_RCV_SYSTEM_ERROR.
h	Signal caller with PROTOCOL_ERROR_RCVD.

DS_RCV_SEND_ERR_SUSP_TERM_REMU

Function: This finite-state machine is signalled when an exception is encountered while receiving a distribution and the immediate exception action is to issue an LU 6.2 Send_Error verb, generate a REMU informing the partner DSU of the report code, and suspend the distribution (if appropriate) or discard it (if suspension is not appropriate).

The FSM first issues an LU 6.2 Send_Error verb (state 1). If the distribution can be restarted via a DCMU, the distribution is retained and the *MU_ID state* changed to SUSPENDED (FSM states 3 and 4). Otherwise, the distribution is discarded and the *MU_ID state* set to TERMINATED (FSM states 3 and 5). A REMU is generated to inform the partner DSU of the exception report code (state 6). This FSM is signalled when

- DS_RCV_RECEIVE_DMU detects a queue, parser or server exception while receiving a DMU.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines or procedures:
  - from "DS_RCV_RECEIVE_DMU" on page 240
    - START
- Signals from lower-level DS_Receive finite-state machines:
  - from "DS_RCV_SUSP_DIST" on page 274
    - DIST_SUSPENDED
    - SUSPEND_FAILED
  - from "DS_RCV_DISCARD_DIST" on page 276
    - DIST_DISCARDED
    - DISCARD_FAILED
  - from "DS_RCV_MU_ID_REGISTRY" on page 280
    - MU_ID_OP_OK
    - MU_ID_OP_NOT_OK
  - from "DS_RCV_REMU_SUSP_TERM" on page 254
    - DS_RCV_SYSTEM_ERROR
    - SUSP_TERM
  - "UPM_EXCEPT_RECOVERY_ACTION" on page 285
    - NOT_RETRIABLE
    - RETRIABLE_WITH_MID_MU
    - RETRIABLE_WITHOUT_MID_MU
- Signals from machines providing common services:
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK
- Signals from LU 6.2 presentation services
  - OK
  - DEALLOCATE_NORMAL
  - RESOURCE_FAILURE

	States							
	RESET	SEND ERR	REC ACT	SUSP DIST	DISC DIST	MU_ID STATE	REMU	CONV FAIL
Inputs	01	02	03	04	05	06	07	08
START	2a	/	/	/	/	/	/	/
OK	/	3b	/	/	/	/	/	/
DEALLOCATE NORMAL	/	8c	/	/	/	/	/	/
RESOURCE FAILURE	/	8c	/	/	/	/	/	/
NOT RETRIABLE	/	/	5d	/	/	/	/	/
RETRIABLE WITH MID MU	/	/	4e	/	/	/	/	/
RETRIABLE WITHOUT MID MU	/	/	5d	/	/	/	/	/
DIST SUSPENDED	/	/	/	6f	/	/	/	/
SUSPEND FAILED	/	/	/	5d	/	/	/	/
DIST DISCARDED	/	/	/	/	6g	/	/	/
DISCARD FAILED	/	/	/	/	6g	/	/	/
MU ID OP OK	/	/	/	/	/	7h	/	/
MU ID OP NOT OK	/	/	/	/	/	1i	/	/
QUEUE OK	/	/	/	/	/	/	1j	/
QUEUE NOT OK	/	/	/	/	/	/	1i	/
SUSP TERM	/	/	/	/	/	/	/	1k

Output Code	Function
a	Signal LU 6.2 presentation services with SEND_ERROR.
b	Signal UPM_EXCEPT_RECOVERY_ACTION with the error code.
c	Signal DS_RCV_REMU_SUSP_TERM with START.
d	Signal DS_RCV_DISCARD_DIST with START.
e	Signal DS_RCV_SUSP_DIST with START.
f	Signal DS_RCV_MU_ID_REGISTRY with SUSPENDED.
g	Signal DS_RCV_MU_ID_REGISTRY with TERMINATED.
h	Build a REMU with the appropriate exception code and enqueue it on the control MU queue by signalling QUEUE_MGR with WRITEQ specifying *queue*(CONTROL_MU_QUEUE).
i	Signal caller with DS_RCV_SYSTEM_ERROR.
j	Signal caller with SEND_ERR_SUSP_TERM_REMU.
k	Signal caller with CONVERSATION_FAILURE.

DS_RCV_REMU_SUSP_TERM

Function: This finite-state machine is signalled whenever an exception is detected while receiving a distribution, and the immediate exception action is to generate a REMU informing the partner of the report code, and to suspend or discard the distribution. The distribution is suspended if the exception is retriable and the distribution can be restarted with a DCMU. Otherwise, the distribution is discarded.

In states 2 and 3, the distribution is suspended and the *MU_ID state* is set to SUSPENDED if the exception is retriable and the distribution can be restarted via a DCMU. Otherwise, the distribution is discarded and the *MU_ID state* set to TERMINATED (FSM states 2 and 4). Finally, a REMU is generated (state 6).

This FSM is signalled when

- A conversation failure is detected during receipt of a DMU or during a DMU's exception processing. The REMU generated here will be sent to the partner by a different instance of DS_Receive, and on a different conversation. (Even though conversation failure is retriable, this FSM handles NOT_RETRIABLE exceptions because the conversation failure might occur during a non-retriable exception's processing. In such a case, the distribution is discarded.)

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines or procedures:
  - from "DS_RCV_RECEIVE_DMU" on page 240
    - START
  - from "DS_RCV_SEND_ERR_SUSP_TERM_REMU" on page 252
    - START
- Signals from lower-level DS_Receive finite-state machines:
  - from "DS_RCV_SUSP_DIST" on page 274
    - DIST_SUSPENDED
    - SUSPEND_FAILED
  - from "DS_RCV_DISCARD_DIST" on page 276
    - DIST_DISCARDED
    - DISCARD_FAILED
  - from "DS_RCV_MU_ID_REGISTRY" on page 280
    - MU_ID_OP_OK
    - MU_ID_OP_NOT_OK
  - from "UPM_EXCEPT_RECOVERY_ACTION" on page 285
    - NOT_RETRIABLE
    - RETRIABLE_WITH_MID_MU
    - RETRIABLE_WITHOUT_MID_MU
- Signals from machines providing common services:
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK

	States					
	RESET	REC ACT	SUSP DIST	DISC DIST	MU_ID STATE	REMU
Inputs	01	02	03	04	05	06
START	2a	/	/	/	/	/
NOT RETRIABLE	/	4b	/	/	/	/
RETRIABLE WITH MID MU	/	3c	/	/	/	/
RETRIABLE WITHOUT MID MU	/	4b	/	/	/	/
DIST SUSPENDED	/	/	5d	/	/	/
SUSPEND FAILED	/	/	4b	/	/	/
DIST DISCARDED	/	/	/	5e	/	/
DISCARD FAILED	/	/	/	5e	/	/
MU ID OP OK	/	/	/	/	6f	/
MU ID OP NOT OK	/	/	/	/	1g	/
QUEUE OK	/	/	/	/	/	1h
QUEUE NOT OK	/	/	/	/	/	1g

Output Code	Function
a	Signal UPM_EXCEPT_RECOVERY_ACTION with the error code.
b	Signal DS_RCV_DISCARD_DIST with START.
c	Signal DS_RCV_SUSP_DIST with START.
d	Signal DS_RCV_MU_ID_REGISTRY with SUSPENDED.
e	Signal DS_RCV_MU_ID_REGISTRY with TERMINATED.
f	Build a REMU with the appropriate exception code and signal QUEUE_MGR with WRITEQ specifying *queue*(CONTROL_MU_QUEUE).
g	Notify operations of the exception condition. Signal caller with REMU_SUSP_TERM.
h	Signal caller with REMU_SUSP_TERM.

DS_RCV_SEND_ERR_CRMU

Function: This finite-state machine is signalled when an exception has been encountered while receiving a distribution and the immediate exception action is to issue an LU 6.2 Send_Error verb to terminate the transmission, and generate a CRMU informing the partner DSU that the *MU_ID state* was already COMPLETED or PURGED. Eventually, DS_Receive will send this CRMU (and any other CMUs waiting to be sent) to the partner. The conversation will be deallocated (if appropriate). This FSM is signalled by DS_RCV_RECEIVE_DMU when an MU whose *MU_ID state* is already COMPLETED or PURGED is received.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines or procedures:
  - from "DS_RCV_RECEIVE_DMU" on page 240
    - MU_ID_COMP_PURG
- Signals from machines providing common services:
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK
- Signals from LU 6.2 presentation services
  - OK
  - ALLOCATION_ERROR
  - RESOURCE_FAILURE

	States			
	RESET	SEND ERR	SEND CRMU	SEND CRMU CONV FAIL
Inputs	01	02	03	04
MU_ID COMP PURG	2a	/	/	/
OK	/	3b	/	/
DEALLOCATE NORMAL	/	4b	/	/
RESOURCE FAILURE	/	4b	/	/
QUEUE OK	/	/	1c	1e
QUEUE NOT OK	/	/	1d	1e

Output Code	Function
a	Signal LU 6.2 presentation services with SEND_ERROR.
b	Build a CRMU with the appropriate exception code and enqueue it on the control MU queue by signalling QUEUE_MGR with WRITEQ specifying *queue*(CONTROL_MU_QUEUE).
c	Signal caller with MU_ID_COMP_PURG.
d	Signal caller with DS_RCV_SYSTEM_ERROR.
e	Signal caller with CONVERSATION_FAILURE.

DS_RCV_RECEIVE_DMU_NO_MU_ID

Function: This finite-state machine is signalled to control the receiving and parsing of a basic-integrity DTMU or DRMU. States 2 and 3 form a loop receiving and parsing the DMU's control information. States 2, 3 and 4 form a loop receiving and storing the server object. State 5 signals DS_RCV_ENQ_SCHED to accept responsibility for the distribution, putting it on the router-director queue, and scheduling DS_Router_Director.

A basic-integrity distribution does not use an *MU_ID* or CMUs to control its transfer from one DSU to the next. Any exception (including exceptions unrelated to the distribution, such as conversation failure) results in the distribution being discarded.

Note: An elective (not shown in this model) allows a REMU to be generated or, receiver-detected errors (i.e., those for which RCV_SIDE_EXCEPT CONVERSATION_FAILURE or PROTOCOL_ERROR are returned to the caller). No *MU_ID* will be included in this REMU.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines:
  - from "DS_RCV_RECEIVE_DMU" on page 240
    - START
- Signals from lower-level DS_Receive finite-state machines:
  - from "DS_RCV_ENQ_SCHED" on page 262
    - ENQ_SCHED_OK
    - ENQ_SCHED_NOT_OK
    - ENQ_SCHED_OK_DEALLOCATE
  - from "DS_RCV_DISCARD_DIST" on page 276
    - DIST_DISCARDED
    - DISCARD_FAILED
- Signals from finite-state machines providing common services:
  - from "RCV_BUFFER_MGR" on page 282
    - OK
    - CONVERSATION_FAILURE
    - PROTOCOL_ERROR
    - DEALLOCATE_NORMAL
    - PROG_ERROR
    - CHANGE_DIRECTION
  - "SERVER_MGR" on page 354
    - OBJECT_OK
    - OBJECT_NOT_OK
    - SPECIFIC_SERVER_EXCEPTION
- Signals from LU 6.2 presentation services
  - OK
  - ALLOCATION_ERROR
  - DEALLOCATE_NORMAL
  - RESOURCE_FAILURE
- Signals from "PARSER" on page 359:
  - PARSE_OK
  - PARSE_OK_OBJECT
  - PARSE_COMPLETE
  - PARSE_NOT_OK

	States										
	RESET	PARS	RCV NEXT	WRIT OBJ	END OBJ	ENQ SCHD	SEND ERR	RCV SIDE ERR	PROG ERR	CONV FAIL	PROT ERR
Inputs	01	02	03	04	05	06	07	08	09	10	11
START	2a	/	/	/	/	/	/	/	/	/	/
PARSE OK	/	3b	/	/	/	/	/	/	/	/	/
PARSE OK OBJECT	/	4c	/	/	/	/	/	/	/	/	/
PARSE COMPLETE	/	5d	/	/	/	/	/	/	/	/	/
PARSE NOT OK	/	7e	/	/	/	/	/	/	/	/	/
OK	/	/	2a	/	/	/	8f	/	/	/	/
PROG ERROR	/	/	9f	/	/	/	/	/	/	/	/
CONVERSATION FAILURE	/	/	10f	/	/	/	/	/	/	/	/
PROTOCOL ERROR	/	/	11f	/	/	/	/	/	/	/	/
DEALLOCATE NORMAL	/	/	10f	/	/	/	10f	/	/	/	/
CHANGE DIRECTION	/	/	11f	/	/	/	/	/	/	/	/
RESOURCE FAILURE	/	/	/	/	/	/	10f	/	/	/	/
OBJECT OK	/	/	/	3b	6g	/	/	/	/	/	/
OBJECT NOT OK	/	/	/	7e	7e	/	/	/	/	/	/
SPECIFIC SERVER EXCEPTION	/	/	/	3b	/	/	/	/	/	/	/
ENQ SCHED OK	/	/	/	/	/	1h	/	/	/	/	/
ENQ SCHED OK DEALLOC	/	/	/	/	/	1i	/	/	/	/	/
ENQ SCHED NOT OK	/	/	/	/	/	7e	/	/	/	/	/
DIST DISCARDED	/	/	/	/	/	/	/	1j	1k	1m	1n
DISCARD FAILED	/	/	/	/	/	/	/	1j	1k	1m	1n

Output Code	Function
a	Signal PARSER with PARSE to parse information received.
b	Signal RCV_BUFFER_MGR with RECEIVE_BUFFER to receive information from LU 6.2.
c	Signal SERVER_MGR with WRITE to write the just-received portion of the server object. SERVER_MGR will issue a Initiate_Write, if appropriate.
d	Signal SERVER_MGR with TERMINATE_WRITE. If no server object has been processed (and therefore no initiate write has been issued), processing continues with OBJECT_OK.
e	Signal LU 6.2 presentation services with SEND_ERROR.
f	Signal DS_RCV_DISCARD_DIST with START to discard the partially-received distribution and its server object (if any).
g	Signal DS_RCV_ENQ_SCHED with ENQ_SCHED to place the distribution on the router-director queue and schedule DS_Router_Director.
h	Signal caller with MU_OK to indicate that a DMU has been completely received.
i	Signal caller with DS_RCV_SYSTEM_ERROR.
j	Signal caller with RCV_SIDE_EXCEPT.
k	Signal caller with SEND_SIDE_EXCEPT to indicate that an error has occurred in DS_Send and that the sender exception protocols should be followed.
m	Signal caller with CONVERSATION_FAILURE.
n	Signal caller with PROTOCOL_ERROR to indicate that a protocol error has occurred in the conversation between DS_Send and DS_Receive.

DS_RCV_ENQ_SCHED

Function: This finite-state machine describes the functional processing in DS_Receive to accept responsibility for a distribution. In states 1-4, it places the distribution on the router-director queue, updates the *MU_ID state* to COMPLETED, and schedules the DS_Router_Director. If any of these operations fails, the others are either not attempted or backed out. State 5 removes the distribution from the mid-MU restart queue.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines:
  - from "DS_RCV_RECEIVE_DMU" on page 240
    - ENQ_SCHED
  - from "DS_RCV_RECEIVE_DMU_NO_MU_ID" on page 259
    - ENQ_SCHED
- Signals from lower-level DS_Receive finite-state machines:
  - "DS_RCV_MU_ID_REGISTRY" on page 280
    - see output code D.
- Signals from finite-state machines providing common services:
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK
  - from "FSM_SCHED_MGR" on page 351
    - SCHED_FUNCTION_OK
    - SCHED_FUNCTION_NOT_OK

	States						
	RESET	**ENQ PEND**	**SCHED PEND**	**ACCEPT**	**DEQ MID MU**	**CRMU**	**ACCEPT**
Inputs	**01**	**02**	**03**	**04**	**05**	**06**	**07**
ENQ SCHED	2a	/	/	/	/	/	/
QUEUE OK	/	3b	/	/	6g	1i	1c
QUEUE NOT OK	/	1c	/	/	6h	1j	1c
SCHED FUNCTION OK	/	/	4d	/	/	/	/
SCHED FUNCTION NOT OK	/	/	7e	/	/	/	/
ATOMIC OPERATION SUCCESSFUL	/	/	/	5f	/	/	/
ATOMIC OPERATION FAILED	/	/	/	7e	/	/	/

Output Code	Function
a	Signal QUEUE_MGR with WRITEQ specifying *queue*(ROUTER_DIRECTOR_QUEUE) to place the distribution on the router-director queue.
b	Signal FSM_SCHED_MGR with START_REQUEST to schedule DS_Router_Director.
c	Signal caller with ENQ_SCHED_NOT_OK to indicate that an error occurred in placing the distribution on the router-director queue, or in scheduling DS_Router_Director. Cleanup of the router-director queue has taken place.
d	As an atomic operation (i.e., all or none of the following operations must succeed), perform the following actions: If the distribution is using *MU_ID*s, then signal DS_RCV_MU_ID_REGISTRY with INSPECT; if its value is IN_TRANSIT, then change the entry to COMPLETE. Signal QUEUE_MGR with RELEASEQ specifying *queue*(ROUTER_DIRECTOR_QUEUE) to remove the in-use mark from the entry on the router-director queue. The entry is then available for processing by DS_Router_Director.
e	Signal QUEUE_MGR with DEQ specifying *queue*(ROUTER_DIRECTOR_QUEUE) to remove the distribution from the router-director queue.
f	Signal QUEUE_MGR with DEQ specifying *queue*(MID_MU_RESTART_QUEUE) to discard the copy of the distribution on the mid-MU restart queue if mid-MU restart is available. If mid-MU restart is not available, processing continues as though QUEUE_MGR returned QUEUE_OK.
g	Build a CRMU specifying the *MU_ID state* as COMPLETE and signal QUEUE_MGR with WRITEQ specifying *queue*(CONTROL_MU_QUEUE) if *MU_ID*s are being used. If *MU_ID*s are not being used, generating the CRMU is optional; if no CRMU is generated, processing continues as though QUEUE_MGR returned QUEUE_OK.
h	Notify operations of the exception condition. Build a CRMU specifying the *MU_ID state* as COMPLETE and signal QUEUE_MGR with WRITEQ specifying *queue*(CONTROL_MU_QUEUE).
i	Signal caller with ENQ_SCHED_OK to indicate that the distribution was successfully placed on the router-director queue and that DS_Router_Director was scheduled.
j	Notify operations of the exception condition. Signal caller with ENQ_SCHED_OK_DEALLOCATE to indicate that the distribution was successfully placed on the router-director queue, that DS_Router_Director was scheduled and that a subsequent system error has occurred that should cause the conversation to be aborted.

DS_RCV_CQMU_HANDLER

Function: This finite-state machine describes the functional processing for DS_Receive receiving a CQMU from its partner DSU. In state 2, the *instance number* is checked. If it is too low, the CQMU is discarded with no further action. If the *instance number* is acceptable, the *MU_ID state* is inspected and used to generate a CRMU. *MU_ID*s lower than the lowest directory entry are considered NOT_RECEIVED, and those higher than the highest directory entry are considered PURGED.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines or procedures:
  - from "DS_RCV_RECEIVING" on page 237
    - START
- Signals from lower-level DS_Receive finite-state machines:
  - from "DS_RCV_MU_ID_REGISTRY" on page 280
    - NOT_RECEIVED
    - IN_TRANSIT
    - SUSPENDED
    - TERMINATED
    - COMPLETE
    - PURGED
    - INSTANCE_NUM_HIGH
    - INSTANCE_NUM_EQUAL
    - INSTANCE_NUM_LOW
    - MU_ID_AGED_OUT
    - MU_ID_UNINITIALIZED
    - MU_ID_NOT_OK
- Signals from machines providing common services:
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK
  - from "SERVER_MGR" on page 354
    - OBJECT_OK
    - OBJECT_NOT_OK

	States				
	RESET	INST NUMB	MU_ID STATE	QUERY BYTE NUM	RETURN
Inputs	01	02	03	04	05
START	2a	/	/	/	/
INSTANCE NUM HIGH	/	3b	/	/	/
INSTANCE NUM EQUAL	/	3b	/	/	/
INSTANCE NUM LOW	/	1c	/	/	/
NOT RECEIVED	/	/	5e	/	/
IN TRANSIT	/	/	5e	/	/
SUSPENDED	/	/	4h	/	/
TERMINATED	/	/	5e	/	/
COMPLETED	/	/	5e	/	/
PURGED	/	/	5e	/	/
MU ID AGED OUT	/	/	5f	/	/
MU ID UNINITIALIZED	/	/	5g	/	/
MU ID OP NOT OK	/	1d	1d	/	/
OBJECT OK	/	/	/	5e	/
OBJECT NOT OK	/	/	/	5i	/
QUEUE OK	/	/	/	/	1c
QUEUE NOT OK	/	/	/	/	1d

Output Code	Function
a	Signal DS_RCV_MU_ID_REGISTRY with INSTANCE_NUMBER to compare the CQMU's *instance number* with the registry's *instance number*. The CQMU's *instance number* is acceptable if it is greater than or equal to the registry's value. If the CQMU's *instance number* is less than the registry's, INSTANCE_NUM_LOW is returned (and the CQMU will be discarded). If the CQMU's *instance number* is greater than the registry's *instance number*, the registry's number is updated to be equal to the CQMU.
b	Signal DS_RCV_MU_ID_REGISTRY with INSPECT.
c	Signal the caller with MU_OK.
d	Signal the caller with MU_NOT_OK.
e	Build a CRMU and signal QUEUE_MGR with WRITEQ specifying *queue*(CONTROL_MU_QUEUE).
f	Build a CRMU and signal QUEUE_MGR with WRITEQ specifying *queue*(CONTROL_MU_QUEUE). The CRMU specifies that the designated *MU_ID state* is PURGED.
g	Build a CRMU and signal QUEUE_MGR with WRITEQ specifying *queue*(CONTROL_MU_QUEUE). The CRMU specifies that the designated *MU_ID state* is NOT_RECEIVED.
h	Signal SERVER_MGR with QUERY_LAST_BYTE_RCVD to get the correct byte number for the CRMU. If the DTMU was not suspenrled in the server object, processing continues as though OBJECT_OK was returned.
i	Build a CRMU and signal QUEUE_MGR with WRITEQ specifying *queue*(CONTROL_MU_QUEUE). The CRMU specifies that the designated *MU_ID state* is TERMINATED.

DS_RCV_SEMU_HANDLER

Function: This finite-state machine describes the functional processing for DS_Receive receiving a SEMU from its partner DSU. In state 2, the *instance number* and the presence of an *MU_ID* are checked. SEMUs with *instance numbers* lower than the *instance number* of the MU_ID registry are considered tardy and simply discarded. A SEMU without an *MU_ID* is logged, and no other action is taken. In state 3, the *MU_ID state* from the registry is inspected, and the appropriate exception actions initiated (states 4, 5, 6 and 7). If the *MU_ID state* is NOT_RECEIVED, the SEMU is informing DS_Receive that the partner will *never* use that *MU_ID*, and the *MU_ID state* is changed to TERMINATED. For SUSPENDED *MU_ID*s, the exception code is examined (state 4) to determine if the sender's exception is retriable or not. If not, the distribution is discarded and the *MU_ID state* changed to TERMINATED. A CRMU reporting the *MU_ID state* is generated as the last exception action.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines or procedures:
  - from "DS_RCV_RECEIVING" on page 237
    - START
- Signals from lower-level DS_Receive finite-state machines:
  - from "DS_RCV_DISCARD_DIST" on page 276
    - DIST_DISCARDED
    - DISCARD_FAILED
  - from "DS_RCV_MU_ID_REGISTRY" on page 280
    - NOT_RECEIVED
    - IN_TRANSIT
    - SUSPENDED
    - TERMINATED
    - COMPLETE
    - PURGED
    - INSTANCE_NUM_HIGH
    - INSTANCE_NUM_EQUAL
    - INSTANCE_NUM_LOW
    - MU_ID_OP_OK
    - MU_ID_OP_NOT_OK
- Signals from machines providing common services:
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK
  - from "SERVER_MGR" on page 354
    - OBJECT_OK
    - OBJECT_NOT_OK
  - from "UPM_EXCEPT_RECOVERY_ACTION" on page 285
    - NOT_RETRIABLE
    - RETRIABLE_WITH_MID_MU

	States							
	RESET	INST NUM	FIND MU_ID	SUSP	QUERY LAST BYTE	DISC DIST	TERM	CRMU
Inputs	01	02	03	04	05	06	07	08
START	2a	/	/	/	/	/	/	/
INSTANCE NUM HIGH	/	3b	/	/	/	/	/	/
INSTANCE NUM EQUAL	/	3b	/	/	/	/	/	/
INSTANCE NUM LOW	/	1c	/	/	/	/	/	/
NO MU ID IN MU	/	1c	/	/	/	/	/	/
NOT RECEIVED	/	/	7e	/	/	/	/	/
IN TRANSIT	/	/	8f	/	/	/	/	/
SUSPENDED	/	/	4g	/	/	/	/	/
TERMINATED	/	/	8f	/	/	/	/	/
COMPLETED	/	/	8f	/	/	/	/	/
PURGED	/	/	8f	/	/	/	/	/
NOT RETRIABLE	/	/	/	6h	/	/	/	/
RETRIABLE WITH MID MU	/	/	/	5i	/	/	/	/
OBJECT OK	/	/	/	/	8f	/	/	/
OBJECT NOT OK	/	/	/	/	6h	/	/	/
DIST DISCARDED	/	/	/	/	/	7e	/	/
DISCARD FAILED	/	/	/	/	/	7e	/	/
MU ID OP OK	/	/	/	/	/	/	8f	/
MU ID OP NOT OK	/	1d	1d	/	/	/	1d	/
QUEUE OK	/	/	/	/	/	/	/	1c
QUEUE NOT OK	/	/	/	/	/	/	/	1d

Output Code	**Function**
a	Signal DS_RCV_MU_ID_REGISTRY with INSTANCE_NUMBER to compare the SEMU's *instance number* with the registry's *instance number*. The SEMU's *instance number* is acceptable if it is greater than or equal to the registry's value. If the SEMU's *instance number* is less than the registry's, INSTANCE_NUM_LOW is returned. If the SEMU's *instance number* is greater than the registry's *instance number*, the registry is updated. If the SEMU contains no *MU_ID*, processing continues with the NO_MU_ID_IN_MU signal.
b	Signal DS_RCV_MU_ID_REGISTRY with INSPECT.
c	SEMUs containing no *MU_ID* are reported to operations. Signal the caller with MU_OK.
d	Signal the caller with MU_NOT_OK.
e	Signal DS_RCV_MU_ID_REGISTRY with TERMINATED.
f	Build a CRMU specifying the *MU_ID's state* and signal QUEUE_MANAGER with WRITEQ specifying *queue*(CONTROL_MU_QUEUE).
g	Signal UPM_EXCEPT_RECOVERY_ACTION with the error code.
h	Signal DS_RCV_DISCARD_DIST with START.
i	Signal SERVER_MGR with QUERY_LAST_BYTE_RCVD

DS_RCV_PRMU_HANDLER

Function: This finite-state machine describes the functional processing for DS_Receive receiving a PRMU from its partner DSU. If the *MU_ID state* in the registry is SUSPENDED, TERMINATED or COMPLETED, it is changed to PURGED as a result of receiving the PRMU. If the *MU_ID state* was SUSPENDED prior to receiving the PRMU, the server object is deleted, and any other resources dedicated to the distribution (e.g. control blocks and mid-MU restart queue entries) may be reclaimed as part of the PRMU processing.

If the *MU_ID state* was NOT_RECEIVED or IN_TRANSIT prior to receiving the PRMU, the *MU_ID state* is not changed and the PRMU ignored. If the *MU_ID state* was already PURGED when the PRMU was received, no actions are taken.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines or procedures:
  - from "DS_RCV_RECEIVING" on page 237
    - START
- Signals from lower-level DS_Receive finite-state machines:
  - from "DS_RCV_DISCARD_DIST" on page 276
    - DIST_DISCARDED
    - DISCARD_FAILED
  - from "DS_RCV_MU_ID_REGISTRY" on page 280
    - SUSPENDED
    - TERMINATED
    - COMPLETE
    - PURGED
    - MU_ID_AGED_OUT
    - MU_ID_OP_OK
    - MU_ID_OP_NOT_OK

	States			
	RESET	MU_ID STATE	CLEANUP DIST	PURGE
Inputs	01	02	03	03
START	2a	/	/	/
NOT RECEIVED	/	3f	/	/
IN TRANSIT	/	3f	/	/
SUSPENDED	/	3b	/	/
TERMINATED	/	4c	/	/
COMPLETED	/	4c	/	/
PURGED	/	1d	/	/
MU ID AGED OUT	/	1d	/	/
MU ID OP OK	/	/	/	1d
MU ID OP NOT OK	/	1e	/	1e
DIST DISCARDED	/	/	4c	/
DISCARD FAILED	/	/	4c	/

Output Code	Function
a	Signal DS_RCV_MU_ID_REGISTRY with INSPECT.
b	Signal DS_RCV_DISCARD_DIST with START.
c	Signal DS_RCV_MU_ID_REGISTRY with PURGE.
d	Signal the caller with MU_OK.
e	Signal the caller with MU_NOT_OK.
f	Notify operations of the exception condition (that is, the unsolicited PRMU). Signal the caller with MU_OK.

DS_RCV_SUSP_DIST

Function: This finite-state machine suspends a distribution which has been partially received. Processing of the distribution may be resumed later, possibly by a different instance of DS_Receive. The mid-MU restart queue entry is released (so that it may be found later by a READQ queue operation), and a Terminate_Write server verb is issued, if necessary.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines or procedures:
  - from "DS_RCV_SUSP_TERM" on page 250
    - START
  - from "DS_RCV_SEND_ERR_SUSP_TERM_REMU" on page 252
    - START
  - from "DS_RCV_REMU_SUSP_TERM" on page 254
    - START
- Signals from machines providing common services:
  - from "SERVER_MGR" on page 354
    - OBJECT_OK
    - OBJECT_NOT_OK
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK

	States			
	RESET	RETRY ACTION	TERM WRITE	RELQ FAILED
Inputs	01	02	03	04
START	2a	/	/	/
QUEUE OK	/	3b	/	/
QUEUE NOT OK	/	4b	/	/
OBJECT OK	/	/	1c	1d
OBJECT NOT OK	/	/	1d	1d

Output Code	Function
a	Signal QUEUE_MGR with RELEASEQ specifying *queue*(MID_MU_RESTART_QUEUE).
b	Signal SERVER_MGR with TERMINATE_WRITE specifying *type*(SUSPEND) to preserve the partially-received server object, if appropriate. If this distribution does not have a server object, processing continues with the QUEUE_OK signal.
c	Signal caller with DIST_SUSPENDED.
d	Signal caller with SUSPEND_FAILED.

DS_RCV_DISCARD_DIST

Function: This finite-state machine describes the functional processing for deleting a partially-received distribution. It is called after an exception has been detected during the distribution's transfer from partner DSU to DS_Receive, and after DS_Receive has determined that restarting the transmission with a DCMU is not possible.

Note: This FSM assumes that the general server is being used to store the server object, and that a later auxiliary operation will copy the server object from the general server to the specific server, if necessary. If, instead, DS_Receive directly stores the server object into the specific server, then DS_Receive signals the server manager with either BACKOUT (if Terminate_Write has already been issued) or with TERMINATE_WRITE specifying ABORT. DS_Receive then queues for the destination agent any specific server report that was returned from server manager, and signals server manager with DECREMENT_OBJECT_LOCK.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines or procedures:
  - from "DS_RCV_RECEIVE_DMU_NO_MU_ID" on page 259
    - START
  - from "DS_RCV_SUSP_TERM" on page 250
    - START
  - from "DS_RCV_SEND_ERR_SUSP_TERM_REMU" on page 252
    - START
  - from "DS_RCV_REMU_SUSP_TERM" on page 254
    - START
  - from "DS_RCV_SEMU_HANDLER" on page 269
    - START
  - from "DS_RCV_PRMU_HANDLER" on page 272
    - START
- Signals from machines providing common services:
  - from "QUEUE_MGR" on page 357
    - QUEUE_OK
    - QUEUE_NOT_OK
  - from "SERVER_MGR" on page 354
    - OBJECT_OK
    - OBJECT_NOT_OK

	States					
	RESET	TERM	DEC	DEQ	TERM FAIL DEC	TERM FAIL DEQ
Inputs	01	02	03	04	05	06
START	2a	/	/	/	/	/
OBJECT OK	/	3b	4c	/	6c	/
OBJECT NOT OK	/	5b	6c	/	6c	/
QUEUE OK	/	/	/	1d	/	1e
QUEUE NOT OK	/	/	/	1e	/	1e

Output Code	Function
a	If appropriate, signal SERVER_MGR with TERMINATE_WRITE terminating restartability of the server object. If a Terminate_Write specifying SUSPEND has been issued previously, then signal SERVER_MGR with TERMINATE_RESTARTABILITY, if appropriate. If no Initiate_Write has opened the server object or no server object exists for this distribution, SERVER_MGR returns OBJECT_OK.
b	Signal SERVER_MGR with DECREMENT_OBJ_LOCK to decrement the DS lock count on the server object. If no server object exists, OBJECT_OK is returned.
c	Signal QUEUE_MGR with DEQ specifying *queue*(MID_MU_RESTART_QUEUE) to remove the partial distribution. If mid-MU restart capability is not being used for this distribution, QUEUE_OK is returned.
d	Signal caller with DIST_DISCARDED.
e	Notify operations of the exception condition. Signal caller with DISCARD_FAILED.

DS_RCV_SEND_CONVERSATION_MGR

Function: This finite-state machine describes the functional processing for sending a buffer to LU 6.2 presentation services using the the Send_Data verb.

This FSM gets control from one of the following:

- Signals from higher-level finite-state machines:
  - from "DS_RCV_SENDING" on page 234
    - SEND_BUFFER
- Signals from LU 6.2 presentation services:
  - ALLOCATION_ERROR
  - DEALLOCATE_ABEND
  - PROG_ERROR
  - SVC_ERROR
  - RESOURCE_FAILURE
  - OK
  - OK_AND_REQUEST_TO_SEND

	States	
	RESET	SEND DATA
Inputs	01	02
SEND BUFFER	2a	/
OK	/	1b
DEALLOCATE ABEND	/	1c
RESOURCE FAILURE	/	1c
PROG ERROR	/	1d
OK AND REQUEST TO SEND	/	1e
SVC ERROR	/	1e
ALLOCATION ERROR	/	1c

Output Code	Function
a	Signal LU 6.2 presentation services with SEND_DATA to send the encoded information to the partner DSU.
b	Signal caller with OK to indicate that the LU 6.2 request was completed successfully.
c	Signal caller with CONVERSATION_FAILURE to indicate that some error has occurred on the conversation between DS_Send and DS_Receive.
d	Signal caller with PROG_ERROR to indicate that the partner DSU has signalled that an error has occurred.
e	Signal caller with PROTOCOL_ERROR to indicate that an error in the protocols has occurred between DS_Send and DS_Receive.

DS_RCV_MU_ID_REGISTRY

This FSM manages DS_Receive's MU_ID registry for a single connection. Its inputs and actions are listed below. All inputs except INSPECT and INSTANCE_NUMBER return either MU_ID_OP_OK or MU_ID_OP_NOT_OK.

- IN_TRANSIT

  This input sets the *MU_ID state* to IN_TRANSIT.

- SUSPENDED

  This input sets the *MU_ID state* to SUSPENDED.

- TERMINATED

  This input sets the *MU_ID state* to TERMINATED.

- COMPLETED

  This input sets the *MU_ID state* to COMPLETED.

- PURGED

  This input sets the *MU_ID state* to PURGED.

- INSPECT

  This input returns the *MU_ID state* of the designated *MU_ID* (NOT_RECEIVED, SUSPENDED, TERMINATED, COMPLETED, or PURGED) or MU_ID_OP_NOT_OK, indicating an error in accessing the MU_ID registry.

- INSTANCE_NUMBER

  Compares a just-received *instance number* with that stored in the MU_ID registry. If the just-received number is less than the registry's value, INSTANCE_NUMBER_LOW is returned. If the values are equal, INSTANCE_NUMBER_EQUAL is returned. If the just-received number is greater than the registry's value, INSTANCE_NUMBER_HIGH is returned, and the registry's *instance number* is replaced by the just-received value. If an MU_ID registry failure prevents the registry from being accessed or updated, MU_ID_OP_NOT_OK is returned.

PREPARSER

This procedure inspects the initial LLID to identify the MU type (DTMU, DCMU, DRMU, CQMU, PRMU, or SEMU), the *MU_ID* and *instance number* (if appropriate). If the initial LL's ID is unrecognized, UNKNOWN_MU is returned. CMUs are parsed completely (by having the preparser signal PARSER if necessary). If an *MU_ID* or *instance number* is improperly specified or missing when required, or if a CMU contains a format exception, then BAD_MU is returned.

FSMs Providing Common Services for FS2 Transport

These FSMs are utilities used by both DS_Send and DS_Receive. They receive data from LU 6.2, detect conversation-idle conditions, and manage the server protocol boundary and queue interface.

The procedures in this class are:

- "RCV_BUFFER_MGR" on page 282

  This FSM issues all LU 6.2 Receive_And_Wait verbs used in the DS transport sublayer. The *what_received* parameter returned from LU 6.2 is simplified to be specific to DS. For example, a *what_received* value of CONFIRM, CONFIRM_SEND, CONFIRM_DEALLOCATE, and SVC_ERROR all represent an invalid usage of the LU 6.2 PB by the partner, and so are collapsed to PROTOCOL_ERROR before returning to the caller.

- "IDLE_DETECTOR" on page 284

  This FSM detects the ″conversation idle″ condition, which indicates that the conversation should be deallocated because DS has no traffic to send over it. The conversation idle condition exists when the partner sent nothing when it was last in LU 6.2 send state, and this DSU has nothing to send currently. In such cases, the conversation is deallocated instead of a Receive_And_Wait being issued. This condition is checked by both DS_Send and DS_Receive. For example, if DS_Send receives the send indication and has nothing to send, it issues a Receive_And_Wait, giving DS_Receive the send indication. If DS_Receive has nothing to send, it deallocates. Conversely, if DS_Receive receives the send indication and has nothing to send, it issues a Receive_And_Wait. If DS_Send has nothing to send, it deallocates.

- "QUEUE_MGR" on page 357

  This procedure is not explicitly specified. It provides a single means for the DS FSMs to access the various queues. QUEUE_MGR performs the specified operation and returns a success or failure indication.

- "SERVER_MGR" on page 354

  This procedure is not explicitly specified. It provides a single means for the DS FSMs to access the server verbs. The verb, with its associated parameters are passed to the server and a success or failure indication is returned.

- "UPM_EXCEPT_RECOVERY_ACTION" on page 285

  This procedure is not explicitly specified. It takes as input parameters an exception condition and the distribution to which the exception relates, and returns a recommended recovery action (such as Not_Retriable, Retriable_Without_Mid_MU, Retriable_With_Mid_MU, or Retriable_Retry_Exhausted).

RCV_BUFFER_MGR

Function: This finite-state machine describes the functional processing to control receiving the data from the partner DSU via LU 6.2. In state 1, a Receive_And_Wait LU 6.2 verb is issued. In state 2, the *what_received* parameter is mapped into a DS-specific return code for the partner. For example, if *what_received* is CONFIRM, CONFIRM_SEND, CONFIRM_DEALLOCATE, or SVC_ERROR, the partner has violated DS's use of the LU 6.2 basic conversation PB, and PROTOCOL_ERROR is returned to the caller.

Deallocate ABEND causes a CONVERSATION_FAILURE to be returned to the caller because, for whatever reason, no conversation exists to transmit information to or receive information from the partner.

This FSM gets control from one of the following:

- Signals from higher-level finite-state machines:
  - from "DS_SEND_RECEIVING" on page 190
    - RECEIVE_BUFFER
  - from "DS_RCV_RECEIVING" on page 237
    - RECEIVE_BUFFER
  - from "DS_RCV_RECEIVE_DMU_NO_MU_ID" on page 259
    - RECEIVE_BUFFER
  - from "DS_RCV_RECEIVE_DMU" on page 240
    - RECEIVE_BUFFER
- Signals from LU 6.2 presentation services:
  - RCV_AND_WAIT_DATA_COMPLETE
  - RCV_AND_WAIT_DATA_INCOMPLETE
  - RCV_AND_WAIT_LL_TRUNCATED
  - RCV_AND_WAIT_SEND
  - RCV_AND_WAIT_CONFIRM_SEND
  - RCV_AND_WAIT_CONFIRM
  - RCV_AND_WAIT_CONFIRM_DEALLOCATE
  - ALLOCATION_ERROR
  - DEALLOCATE_NORMAL
  - DEALLOCATE_ABEND
  - PROG_ERROR
  - SVC_ERROR
  - RESOURCE_FAILURE

	States	
	RESET	RCV
Inputs	01	02
RECEIVE BUFFER	2a	/
RCV AND WAIT DATA COMPLETE	/	1b
RCV AND WAIT DATA INCOMPLETE	/	1b
RCV AND WAIT LL TRUNCATED	/	-a
RCV AND WAIT SEND	/	1c
RCV AND WAIT CONFIRM SEND	/	1d
RCV AND WAIT CONFIRM	/	1d
RCV AND WAIT CONFIRM DEALLOCATE	/	1d
ALLOCATION ERROR	/	1e
DEALLOCATE NORMAL	/	1f
DEALLOCATE ABEND	/	1e
PROG ERROR	/	1g
SVC ERROR	/	1d
RESOURCE FAILURE	/	1e

Output Code	Function
a	Signal LU 6.2 presentation services with RECEIVE_AND_WAIT to indicate to the LU that data can be accepted.
b	Signal caller with OK to indicate that the LU 6.2 request completed successfully.
c	Signal caller with CHANGE DIRECTION.
d	Signal caller with PROTOCOL_ERROR to indicate that the partner has violated the DS protocols.
e	Signal caller with CONVERSATION_FAILURE to indicate that an error occurred on the conversation between DS_Send and DS_Receive.
f	Signal caller with DEALLOCATE_NORMAL. Receiving a DEALLOCATE_NORMAL in all except the first buffer of any MU is a protocol error, but is treated as a conversation failure because the conversation resource is no longer available.
g	Signal caller with PROG_ERROR to indicate that the partner has encountered an error.

IDLE_DETECTOR

Function: This finite-state machine describes the functional processing for detecting that a conversation is idle and should be deallocated. An idle condition exists when the partner DSU sent nothing when it last had the send indication, and the local DSU has nothing to send. Either DS_Send or DS_Receive may deallocate the conversation.

This FSM gets control from one of the following:

- Signals from higher-level finite-state machines or procedures:
  - from "DS_SEND_MANAGER" on page 154
    - CHANGE_DIRECTION
  - from "DS_SEND_BUILD_SEND_DMU" on page 163
    - SOMETHING_SENT
  - from "DS_SEND_SEND_DMU_NO_MU_ID" on page 181
    - SOMETHING_SENT
  - from "DS_SEND_SEND_CONTROL_MU" on page 188
    - SOMETHING_SENT
  - from "DS_SEND_RECEIVING" on page 190
    - SOMETHING_RECEIVED
  - from "DS_RCV_MANAGER" on page 232
    - CHANGE_DIRECTION
  - from "DS_RCV_SENDING" on page 234
    - SOMETHING_SENT
  - from "DS_RCV_RECEIVING" on page 237
    - SOMETHING_RECEIVED

	States	
	RESET	PEND
Inputs	01	02
CHANGE DIRECTION	2a	1b
SOMETHING SENT	-	1
SOMETHING RECEIVED	-	1

Output Code	Function
a	Signal caller with CHANGE_DIRECTION.
b	Signal caller with DEALLOCATE_FLUSH.

UPM_EXCEPT_RECOVERY_ACTION

This procedure is not explicitly specified. Its input parameters include an exception condition, the SENDER_RETRY_ACTION from the REMU (if signalled from DS_Send) and the distribution to which the exception relates. The exception itself is classified as ″retriable″ or ″not retriable″ (see “Characteristics of Exception Conditions” on page 407). If the exception is retriable, the distribution′s retry count may or may not be exhausted. If the exception is retriable and the retry count has not been exhausted, DS_Send or DS_Receive may or may not be able to restart transmitting the distribution with a DCMU. This procedure′s return codes are:

- NOT_RETRIABLE

  The exception is not retriable. In this case, questions of retry counts and restarting with a DCMU are irrelevant. The distribution will not be retried, but will be terminated (and a DRMU generated, as appropriate). The *MU_ID* will be purged.

- RETRIABLE_WITH_MID_MU

  The exception is retriable, the retry count has not been exhausted, and the distribution can be restarted via a DCMU.

- RETRIABLE_WITHOUT_MID_MU

  The exception is retriable, the retry count has not been exhausted, but the distribution cannot be restarted with a DCMU. In this case, the *MU_ID* will be purged, and the distribution will be retransmitted from the beginning with a new *MU_ID*.

- RETRIABLE_RETRY_EXHAUSTED

  The exception is retriable, but DS_Send′s retry count for this distribution has been exhausted. The distribution will not be retried, but will be terminated (and a DRMU generated, as appropriate). The *MU_ID* will be purged.

Distribution Transport Sublayer—Format Set 1

DS Send FSMs

DS Send Overview

DS_Send is shown here as a set of machines that send FS1 DMUs across an LU 6.2 conversation to an instance of DS_Receive in an adjacent DSU. The execution of these machines results in all the DS FS1 protocols for sending DMUs, and all the exception protocols for both sender- and receiver-detected exceptions. There are several points to note about these protocols.

- Deallocate *type*(SYNC_LEVEL) is not issued by the formal model, although product implementations may elect to do so.

- The number of Send_Data verbs issued to LU 6.2 in order to transmit a DS MU is an implementation optimization choice. It will be dependent on the buffer sizes at the nodes at which the DSU resides.

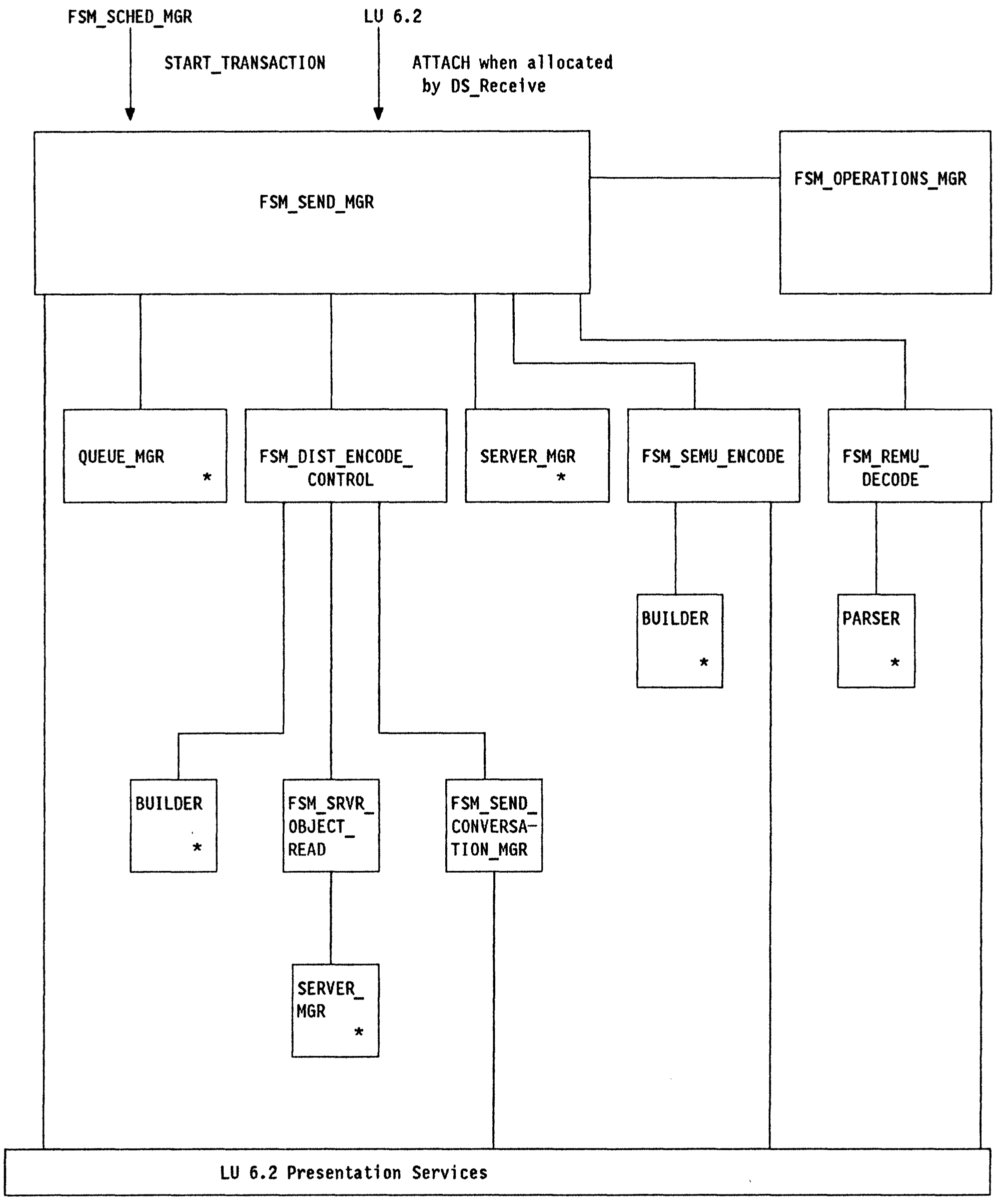
- Each machine that issues verbs to LU 6.2 is listed below, with a summary of its function. All DS implementations follow the protocols embodied in these machines.
  - FSM_SEND_MGR—controls allocation and deallocation of the conversation resource for DS_Send. It handles an Attach from DS_Receive as well as sending an Attach to DS_Receive. It also issues the proper deallocation type for all situations, both exception and normal.
  - DS_SEND_CONVERSATION_MGR—Controls the normal sending of data and the Confirm request. It reports exception conditions to higher-level FSMs for proper exception handling protocols and conversation deallocation.
  - FSM_SEMU_ENCODE—Controls the exception protocols for exceptions detected by the local conversation partner (DS_Send), and handles any further exceptions that may occur during this processing.
  - FSM_REMU_DECODE—Controls the receiving, via LU 6.2, of the Receiver Exception Message Unit (REMU) from the remote conversation partner, and handles any further exceptions that may occur during this processing.

Although the structure of these machines is peculiar to the formal model, all DS implementations generate these protocols. In addition, all DS implementations generate an asynchronous report in the case of nonretriable exceptions.

Program Structure

These FSMs are structured so that a manager machine (FSM_SEND_MGR) both allocates and deallocates the conversation. Once the conversation is started, machines lower in the hierarchy sequence the functions necessary to send a distribution coded in DMU format. The manager machine receives reports on the progress of the transmission. If exceptions are detected, the manager machine causes one of two exception protocol machines to start. The exception protocol machines report on the progress of the exception information being transferred. If an exception occurs during an exception protocol, the manager machine takes appropriate action.

As each piece of the DMU is encoded, FSM_SEND_CONVERSATION_MGR is signalled to issue the appropriate LU 6.2 verb to send the data across the conversation. The manager machine and the exception protocol machines issue their own LU 6.2 verbs. The conversation state is kept in LU 6.2 presentation services. The state of the protocol is kept by a combination of machines and states in DS.



* indicates those finite-state machines not formally specified.

For more details regarding the finite-state machines, see:

- "FSM_SEND_MGR" on page 288
- "FSM_REMU_DECODE" on page 306
- "FSM_OPERATIONS_MGR" on page 342
- "FSM_SRVR_OBJECT_READ" on page 300
- "FSM_DIST_ENCODE_CONTROL" on page 296
- "FSM_SEND_CONVERSATION_MGR" on page 302
- "FSM_SEMU_ENCODE" on page 304
- LU 6.2 presentation services
- "FSM_SCHED_MGR" on page 351
- "PARSER" on page 359
- "SERVER_MGR" on page 354
- "BUILDER" on page 359
- "QUEUE_MGR" on page 357

Figure 47. DS_Send FSM Hierarchy

FSM_SEND_MGR

This machine sequences the operations required to send DMUs from DS_Send at one DSU on an LU 6.2 conversation to DS_Receive at another DSU. The request for this function is from DS_Router_Director, the local operator, or DS_Receive.

The sequences of functions controlled by FSM_SEND_MGR are:

- Allocation of the conversation to DS_Receive, or receiving the indication to send if DS_Send is attached by DS_Receive.
- Accessing the next available queue entry from the next-DSU queues in priority order.
- Encoding the distribution in DMU format.
- Sending the DMU on the LU 6.2 conversation using DS protocols.
- Taking the entry from the next-DSU queue when DS_Receive has accepted responsibility for it.
- Decrementing the DS lock count placed on the server object by presentation services or DS_Receive.
- Deallocation of the conversation when the queues are empty or an exception has occurred.

Processing stops when any of the following occurs:

- The conversation cannot be allocated, or DS_Send does not receive the indication to send, if DS_Receive performs the allocation.
- An exception occurs while the queue is being accessed.
- The queues are held by another instance of DS_Send.
- DS_Receive encounters an exception and notifies DS_Send.
- DS_Send encounters an exception while building the DMU or accessing the server object.
- An exception on the conversation occurs.

For exceptions detected by DS_Send, FSM_SEND_MGR signals machines to pass exception information to DS_Receive. For exceptions detected by DS_Receive, FSM_SEND_MGR signals machines to receive exception information from DS_Receive. For all exceptions, FSM_SEND_MGR performs the appropriate deallocation, and signals FSM_OPERATIONS_MGR for logging, operator messages, setting of queue control, and generating asynchronous report, if appropriate.

The FSM_SEND_MGR has three key states that determine DS actions in exception situations detected by the sending side of the conversation. In the following text, "local conversation partner" refers to the DSU sending the DMU, and "remote conversation partner" refers to the DSU receiving the DMU.

FSM_SEND_MGR—ENC State: In this state, the local conversation manager waits for a report on the encoding and sending of the DMU. The signals received in this state have the following meanings:

- DMU_ABORT—No data has been sent so far on the conversation. An exception was detected on the first call to the builder. This is neither an exception in the distribution, nor an exception in the conversation. DS issues an LU 6.2 Deallocate *type*(FLUSH) verb to terminate the conversation.
- SEND_SIDE_EXCEPT—Another machine in the hierarchy detected an exception during the encoding of the DMU or the reading of the server object. This is a purely local exception, not a conversation exception. In this case, FSM_SEND_MGR signals another machine, FSM_SEMU_ENCODE, to handle the DS exception protocol. This includes informing the conversation partner of the exception and transmitting a SEMU.
- RCV_SIDE_EXCEPT—LU 6.2 PS returned a PROG_ERROR_PURGING on a verb issued by the local conversation partner, indicating that the remote conversation partner has detected an exception. FSM_SEND_MGR signals another machine, FSM_REMU_DECODE, to handle the exception protocol, which includes receiving an REMU from the remote conversation partner.
- CONVERSATION_FAILURE—LU 6.2 PS returns one of the following return codes on a verb that DS issued:
  - ALLOCATION_ERROR—A conversation could not be allocated, but DS has begun to pass records to LU 6.2 PS.
  - DEALLOCATE_ABEND_PROG—The remote conversation partner has detected an exception preventing further communication, and has issued a Deallocate *type*(ABEND_PROG), or the remote LU has signalled a remote program abend condition.
  - RESOURCE_FAILURE—The session has been lost.

  In each of these cases, the conversation no longer exists, and DS issues a Deallocate *type*(LOCAL) to terminate the local side of the conversation.

FSM_SEND_MGR—SND_EXPT State: In this state, FSM_SEND_MGR is waiting for a report on the progress of the exception protocol following the detection of an exception at the local DSU.

- SEND_SIDE_EXCEPT— FSM_SEMU_ENCODE has encountered an exception in encoding the SEMU. The exception protocol cannot be completed, and DS terminates the conversation by issuing Deallocate *type*(ABEND_PROG).
- RCV_SIDE_EXCEPT—The local and remote conversation partners have both issued Send_Error verbs. The remote conversation partner purges everything received from the local conversation partner, including the FMH-7 from the local's Send_Error verb. The result is that the remote conversation partner is in send state when the exchange is complete. LU 6.2 PS returned a PROG_ERROR_PURGING return code in reply to the Send_Error or in reply to the Send_Data verb that the local conversation partner issued. FSM_SEND_MGR signals another machine, FSM_REMU_DECODE, to handle the exception protocol, which includes receiving an REMU from the remote conversation partner.
- CONVERSATION_FAILURE—LU 6.2 PS returns one of the following return codes on a verb that DS issued:

  - ALLOCATION_ERROR—A conversation could not be allocated, but DS has begun to pass records to LU 6.2 PS.
  - DEALLOCATE_ABEND_PROG—The remote conversation partner has detected an exception preventing further communication, and has issued a Deallocate *type*(ABEND_PROG), or the remote LU has signalled a remote program abend condition.
  - RESOURCE_FAILURE—The session has been lost.

  In each of these cases, the conversation no longer exists, and DS issues a Deallocate *type*(LOCAL) to terminate the local side of the conversation.

- ERP_COMPLETE— FSM_SEMU_ENCODE has completed the exception protocol successfully. FSM_SEND_MGR closes the conversation by issuing Deallocate *type*(FLUSH).

FSM_SEND_MGR—RCV_EXPT State: In this state, the FSM_SEND_MGR is waiting for a report on the progress of the exception protocol following the detection of an exception at the remote conversation partner.

- SEND_SIDE_EXCEPT— FSM_REMU_DECODE has encountered an exception in parsing the REMU. The rest of the REMU is ignored, and the remote conversation partner has terminated the conversation normally. The FSM_SEND_MGR issues a Deallocate *type*(LOCAL).
- PROTOCOL_ERROR—LU 6.2 PS returned a PROG_ERROR_TRUNC, PROG_ERROR_NO_TRUNC, *what_received*(SEND), or *what_received*(CONFIRM), to a Receive_And_Wait verb that FSM_REMU_DECODE has issued. These return codes indicate that the partner DSU has issued a sequence of LU 6.2 verbs that, although legitimate under LU 6.2 rules of usage, is not legitimate under DS rules of usage.

  In addition, after the REMU has been received completely, *what_received*(DATA) becomes an improper DS usage of LU 6.2. The local conversation partner must assume that reliable communication cannot continue with the remote conversation partner. The FSM_SEND_MGR issues a Deallocate *type*(ABEND_PROG) to terminate the conversation abnormally.
- CONVERSATION_FAILURE—LU 6.2 PS returns one of the following return codes on a verb that DS issued:
  - DEALLOCATE_ABEND_PROG—The remote conversation partner has detected an exception preventing further communication, and has issued a Deallocate *type*(ABEND_PROG), or the remote LU has signalled a remote program abend condition.
  - RESOURCE_FAILURE—The session has been lost.

  In each of these cases, the conversation no longer exists, and DS issues a Deallocate *type*(LOCAL) to terminate the local side of the conversation. If LU 6.2 PS returns a Deallocate NORMAL before the REMU is received completely, it is an improper DS usage of LU 6.2, but because the conversation no longer exists, FSM_REMU_DECODE signals a CONVERSATION_FAILURE to the FSM_SEND_MGR. This exception is still reported and logged as an improper DS usage of LU 6.2.

- ERP_COMPLETE— FSM_REMU_DECODE has completed the exception protocol successfully. FSM_SEND_MGR closes the conversation by issuing Deallocate *type*(LOCAL).

Function: This finite-state machine describes the functional processing for the send manager of DS_Send. For a further description, see "FSM_SEND_MGR" on page 288.

This FSM gets control from one of the following:

- Signals from finite-state machines providing common services:
  - START_TRANSACTION from FSM_SCHED_MGR
  - OPERATIONS_COMPLETE from FSM_OPERATIONS_MGR
  - OBJECT_OK from SERVER_MGR
  - OBJECT_NOT_OK from SERVER_MGR
- Signals from LU 6.2 presentation services:
  - ATTACH from LU 6.2 when DS_Receive issues Allocate
  - OK from Allocate, Deallocate FLUSH, Deallocate LOCAL, Deallocate ABEND
  - ALLOC_PARAM_ERROR from Allocate
  - ALLOCATION_ERROR from Allocate, Receive_And_Wait
  - RCV_AND_WAIT_SEND from Receive_And_Wait
  - RCV_AND_WAIT_CONFIRM from Receive_And_Wait
  - RCV_AND_WAIT_CONFIRM_DEALLOCATE from Receive_And_Wait
  - RCV_AND_WAIT_DATA_COMPLETE from Receive_And_Wait
  - RCV_AND_WAIT_DATA_INCOMPLETE from Receive_And_Wait
  - RCV_AND_WAIT_LL_TRUNCATED from Receive_And_Wait
  - DEALLOCATE_NORMAL from Receive_And_Wait
  - DEALLOCATE_ABEND from Receive_And_Wait
  - PROG_ERROR from Receive_And_Wait
  - RESOURCE_FAILURE from Receive_And_Wait
- Signals from lower-level DS_Send machines:
  - QUEUE_EMPTY from QUEUE_MGR
  - QUEUE_OK from QUEUE_MGR
  - QUEUE_NOT_OK from QUEUE_MGR
  - DMU_COMPLETE from FSM_DIST_ENCODE_CONTROL
  - DMU_ABORT from FSM_DIST_ENCODE_CONTROL
  - SEND_SIDE_EXCEPT
    - from FSM_DIST_ENCODE_CONTROL
    - from FSM_SEMU_ENCODE
    - from FSM_REMU_DECODE
  - RCV_SIDE_EXCEPT from FSM_DIST_ENCODE_CONTROL, FSM_SEMU_ENCODE
  - PROTOCOL_ERROR from FSM_SEMU_ENCODE, FSM_REMU_DECODE
  - CONVERSATION_FAILURE
    - from FSM_DIST_ENCODE_CONTROL
    - from FSM_SEMU_ENCODE
    - from FSM_REMU_DECODE
  - ERP_COMPLETE from FSM_SEMU_ENCODE, FSM_REMU_DECODE

	States												
	RST	ALLC	RCV	RDQ	ENC	DEQ	DEC LCK	SND EXPT	RCV EXPT	MU EXPT	NMU EXPT	OPS	END
Inputs	01	02	03	04	05	06	07	08	09	10	11	12	13
START TRANSACTION	2a	/	/	/	/	/	/	/	/	/	/	/	/
ATTACH	3b	/	/	/	/	/	/	/	/	/	/	/	/
OK	/	4c	/	/	/	/	/	/	/	12g	12d	/	1
ALLOC PARAM ERROR	/	12g	/	/	/	/	/	/	/	/	/	/	/
ALLOCATION ERROR	/	10e	/	/	/	/	/	/	/	/	/	/	/
RCV AND WAIT SEND	/	/	4c	/	/	/	/	/	/	/	/	/	/
RCV AND WAIT CONFIRM	/	/	11f	/	/	/	/	/	/	/	/	/	/
RCV AND WAIT CONFIRM DEALLOCATE	/	/	11f	/	/	/	/	/	/	/	/	/	/
RCV AND WAIT DATA COMPLETE	/	/	11f	/	/	/	/	/	/	/	/	/	/
RCV AND WAIT DATA INCOMPLETE	/	/	11f	/	/	/	/	/	/	/	/	/	/
RCV AND WAIT LL TRUNCATED	/	/	11f	/	/	/	/	/	/	/	/	/	/
DEALLOC NORMAL	/	/	11e	/	/	/	/	/	/	/	/	/	/
DEALLOC ABEND	/	/	11e	/	/	/	/	/	/	/	/	/	/
PROG ERROR	/	/	11f	/	/	/	/	/	/	/	/	/	/
RESOURCE FAILURE	/	/	11e	/	/	/	/	/	/	/	/	/	/
QUEUE EMPTY	/	/	/	13h	/	/	/	/	/	/	/	/	/
QUEUE OK	/	/	/	5i	/	7n	/	/	/	/	/	/	/
QUEUE NOT OK	/	/	/	11h	/	11h	/	/	/	/	/	/	/
DMU COMPLETE	/	/	/	/	6m	/	/	/	/	/	/	/	/
DMU ABORT	/	/	/	/	10h	/	/	/	/	/	/	/	/
SEND SIDE EXCEPT	/	/	/	/	8j	/	/	10f	10e	/	/	/	/
RCV SIDE EXCEPT	/	/	/	/	9k	/	/	9k	/	/	/	/	/
PROTOCOL ERROR	/	/	/	/	/	/	/	/	10f	/	/	/	/
CONVERSATION FAILURE	/	/	/	/	10e	/	/	10e	10e	/	/	/	/
ERP COMPLETE	/	/	/	/	/	/	/	10h	10e	/	/	/	/
OPERATIONS COMPLETE	/	/	/	/	/	/	/	/	/	/	/	1	/
OBJECT OK	/	/	/	/	/	/	4c	/	/	/	/	/	/
OBJECT NOT OK	/	/	/	/	/	/	11h	/	/	/	/	/	/

Output Code	Function
a	Signal LU 6.2 presentation services with ALLOCATE to establish the conversation with DS_Receive.
b	Signal LU 6.2 presentation services with GET_ATTRIBUTES and RECEIVE_AND_WAIT to determine the LU name, mode name, and to receive the indication from the partner to send.
c	Signal QUEUE_MGR with READQ to retrieve the appropriate next-DSU queue entry to be selected based on priority of traffic.
d	Signal FSM_OPERATIONS_MGR with LOG_MESSAGE_EXCEPT for logging and operator messages. No report is requested because no MU was involved or DS_Receive had already accepted responsibility for the distribution.
e	Signal LU 6.2 presentation services with DEALLOCATE_LOCAL to deallocate the conversation locally.
f	Signal LU 6.2 presentation services with DEALLOCATE_ABEND to deallocate the conversation abnormally.
g	Signal FSM_OPERATIONS_MGR with SEND_MU_EXCEPT to log, generate report if appropriate, set queue control, and for operator messages.
h	Signal LU 6.2 presentation services with DEALLOCATE specifying FLUSH to signal the partner that no more DMUs will be sent.
i	Signal FSM_DIST_ENCODE_CONTROL with ENCODE to control the building and the sending of the DMU to DS_Receive.
j	Signal FSM_SEMU_ENCODE with ENCODE_SEMU to control the exception protocols for an exception encountered by DS_Send.
k	Signal FSM_REMU_DECODE with DECODE_REMU to control the receipt of the REMU that resulted from an exception encountered by DS_Receive.
m	Signal QUEUE_MGR with DEQ to remove the entry from the next-DSU queue following successful processing of the entry.
n	Signal SERVER_MGR with DECREMENT_OBJ_LOCK to decrement the DS lock count on the server object. DS_Receive has accepted responsibility for the distribution. If no object exists, SERVER_MGR returns OBJECT_OK.

FSM_DIST_ENCODE_CONTROL

Controls the building of the distribution in DMU format, and the sending of the DMU on the conversation to DS_Receive. The request for this function is from FSM_SEND_MGR.

This machine signals machines to build and send the DMU. Upon successful completion of building and sending the DMU, this machine signals the FSM_SEND_CONVERSATION_MGR to send the Confirm to DS_Receive. When DS_Receive sends Confirmed, this machine notifies FSM_SEND_MGR that responsibility for the distribution has been accepted by DS_Receive. Exceptions in encoding are reported by the builder. If the builder returns BUILD_NOT_OK on the first call, this machine signals DMU_ABORT to "FSM_SEND_MGR" on page 288. If the builder returns BUILD_NOT_OK on any subsequent call, this machine signals SEND_SIDE_EXCEPT to FSM_SEND_MGR. Exceptions in DS_Receive receiving the DMU result in RCV_SIDE_EXCEPT to FSM_SEND_MGR, and exceptions on the conversation result in CONVERSATION_FAILURE to FSM_SEND_MGR. This machine passes the available exception information to the signalling machine. For a further description of these exceptions, refer to "FSM_SEND_MGR—ENC State" on page 288.

Function: This finite-state machine describes the functional processing for controlling the encoding and sending of the DMU. For a further description, see "FSM_DIST_ENCODE_CONTROL" on page 296.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines:
  - ENCODE from FSM_SEND_MGR
- Signals from lower-level DS_Send finite-state machines:
  - OBJECT_OK from FSM_SRVR_OBJECT_READ
  - NO_OBJECT_EXISTS from FSM_SRVR_OBJECT_READ
  - OBJECT_EOD from FSM_SRVR_OBJECT_READ
  - OBJECT_NOT_OK from FSM_SRVR_OBJECT_READ
  - OK from FSM_SEND_CONVERSATION_MGR
  - RCV_SIDE_EXCEPT from FSM_SEND_CONVERSATION_MGR
  - CONVERSATION_FAILURE from FSM_SEND_CONVERSATION_MGR
- Signals from BUILDER
  - BUILD_OK
  - BUILD_OK_GET_OBJECT
  - BUILD_COMPLETE
  - BUILD_NOT_OK

	States										
	RESET	BLD FIRST	SND	BLD	SND GET OBJ	OBJ PND	SND LST	CON-FIRM	RCV ERP	CNV ERP	SND ERP
Inputs	01	02	03	04	05	06	07	08	09	10	11
ENCODE	2a	/	/	/	/	/	/	/	/	/	/
OK	/	/	4a	/	6f	/	8g	1h	/	/	/
RCV SIDE EXCEPTION	/	/	9e	/	9e	/	9e	9e	/	/	/
CONVERSATION FAILURE	/	/	10e	/	10e	/	10e	10e	/	/	/
BUILD OK	/	3b	/	3b	/	/	/	/	/	/	/
BUILD OK GET OBJECT	/	5b	/	5b	/	/	/	/	/	/	/
BUILD COMPLETE	/	7b	/	7b	/	/	/	/	/	/	/
BUILD NOT OK	/	1c	/	11e	/	/	/	/	/	/	/
OBJECT OK	/	/	/	/	/	4a	/	/	1j	1k	1d
NO OBJECT EXISTS	/	/	/	/	/	4i	/	/	1j	1k	1d
OBJECT EOD	/	/	/	/	/	4i	/	/	/	/	/
OBJECT NOT OK	/	/	/	/	/	1d	/	/	1j	1k	1d

Output Code	Function
a	Signal BUILDER with BUILD to build the DMU information.
b	Signal FSM_SEND_CONVERSATION_MGR with SEND_BUFFER to send the DMU information to LU 6.2 PS.
c	Signal FSM_SEND_MGR with DMU_ABORT to indicate that an exception has occurred in building the DMU. No portion of the DMU has been sent; therefore, normal send side exception processing does not take place.
d	Signal FSM_SEND_MGR with SEND_SIDE_EXCEPT to indicate that an exception has occurred in the encoding process and that the sender exception protocols should be followed.
e	Signal FSM_SRVR_OBJECT_READ with CLEANUP_OBJECT to terminate any access to the object, if one exists.
f	Signal FSM_SRVR_OBJECT_READ with READ_OBJECT to read the object and perform any initialization for reading, if not yet performed.
g	Signal FSM_SEND_CONVERSATION_MGR with END_DMU to indicate that the Confirm should be sent to DS_Receive.
h	Signal FSM_SEND_MGR with DMU_COMPLETE to indicate that the DMU has been encoded and sent to DS_Receive. Also, the Confirm has been sent to DS_Receive and OK to Confirm has been received.
i	Signal BUILDER with END_OBJECT to indicate that the server has returned EOD and the last segment of the object should be built, or if no object exists the length of the data will be 0.
j	Signal FSM_SEND_MGR with RCV_SIDE_EXCEPT to indicate that an exception has occurred in DS_Receive and that the receiver exception protocols should be followed.
k	Signal FSM_SEND_MGR with CONVERSATION_FAILURE to indicate that an exception has occurred in the conversation between DS_Send and DS_Receive.

FSM_SRVR_OBJECT_READ

This machine controls the reading of the server object. The request for this function is from FSM_DIST_ENCODE_CONTROL signalling READ_OBJECT or CLEANUP_OBJECT, and passing the descriptor of the server object, the origin server name, and parameters (if the DSU is the origin DSU).

For an input signal of READ_OBJECT, this machine issues Read to SERVER_MGR. For an input signal of CLEANUP_OBJECT prior to a successful Initiate_Read operation, this machine signals OBJECT_OK to the calling machine. For any subsequent input signal of CLEANUP_OBJECT, this machine issues Terminate_Read to SERVER_MGR and returns OBJECT_NOT_OK.

Exceptions can occur in accessing the server object. For any exception, if an Initiate_Read verb has been issued, then a Terminate_Read verb must be issued to close the access path. In case of an exception from the server in attempting to close the access path, FSM_SRVR_OBJECT_READ should signal FSM_OPERATIONS_MGR with MESSAGE_TO_OPERATOR. This machine does not show that signal.

Exceptions in accessing the server object result in an OBJECT_NOT_OK signal to FSM_DIST_ENCODE_CONTROL and, in turn, a SEND_SIDE_EXCEPT signal to FSM_SEND_MGR. This machine passes the available exception information to FSM_DIST_ENCODE_CONTROL. The processing performed for this exception is described in "FSM_SEND_MGR—ENC State" on page 288.

Function: This finite-state machine describes the functional processing for reading the server object. For a further description, see "FSM_SRVR_OBJECT_READ" on page 300.

This FSM gets control from one of the following:

- Signals from higher-level DS_Send finite-state machines:
  - READ_OBJECT from FSM_DIST_ENCODE_CONTROL
  - CLEANUP_OBJECT from FSM_DIST_ENCODE_CONTROL
- Signals from machines providing common services:
  - OBJECT_OK from SERVER_MGR
  - OBJECT_EOD from SERVER_MGR
  - OBJECT_NOT_OK from SERVER_MGR
  - NO_OBJECT_EXISTS from SERVER_MGR

	States				
	RESET	**READ**	**SEND PEND**	**TRM READ**	**ERP**
Inputs	**01**	**02**	**03**	**04**	**05**
READ OBJECT	2b	/	2b	/	/
CLEANUP OBJECT	1a	/	7e	/	/
NO OBJECT EXISTS	/	1c	/	/	/
OBJECT OK	/	3a	/	1f	1d
OBJECT EOD	/	4e	/	/	/
OBJECT NOT OK	/	5e	/	1d	1d

Output Code	**Function**
a	Signal FSM_DIST_ENCODE_CONTROL with OBJECT_OK to indicate that the portion of the object has been read. If no object exists and the signal is CLEANUP_OBJECT, then OBJECT_OK is signalled.
b	Signal SERVER_MGR with READ to read a portion of the object, performing any initialization, if necessary. If no server object exists, then NO_OBJECT_EXISTS is returned.
c	Signal FSM_DIST_ENCODE_CONTROL with NO_OBJECT_EXISTS to indicate that there is no object to be sent.
d	Signal FSM_DIST_ENCODE_CONTROL with OBJECT_NOT_OK to indicate that an exception has occurred during the accessing of the object.
e	Signal SERVER_MGR with TERMINATE_READ to terminate any access to the object.
f	Signal FSM_DIST_ENCODE_CONTROL with OBJECT_EOD to indicate that the object has been read to completion, and access to the object has been terminated.

FSM_SEND_CONVERSATION_MGR

This machine issues the Send_Data and Confirm verbs to LU 6.2 presentation services, and classifies the conversation exceptions. The request for the send function is from FSM_DIST_ENCODE_CONTROL signalling SEND_BUFFER. The request to send Confirm is from FSM_DIST_ENCODE_CONTROL signalling END_DMU.

Figure 48 classifies the possible exceptions returned from LU 6.2 PS.

Return Code	Reported as	Explanation
ALLOCATION ERROR	CONVERSATION_FAILURE	Conversation was never allocated.
DEALLOCATE ABEND PROG	CONVERSATION_FAILURE	DS_Receive or the remote LU has terminated the conversation abnormally.
PROG ERROR PURGING	RCV_SIDE_ERROR	DS_Receive issued a Send_Error verb to signal error.
RESOURCE FAILURE	CONVERSATION FAILURE	Conversation was lost because the session was lost.

Figure 48. Classification of NOT-OK Return Codes by FSM_SEND_CONVERSATION_MGR.

Function:	This finite-state machine describes the functional processing for interacting with each encode machine for the sending of the DMU to DS_Receive. For a further description, see "FSM_SEND_CONVERSATION_MGR" on page 302. This FSM gets control from one of the following: • Signals from higher-level DS_Send finite-state machines: – SEND_BUFFER from FSM_DIST_ENCODE_CONTROL – END_DMU from FSM_DIST_ENCODE_CONTROL • Signals from LU 6.2 presentation services: – ALLOCATION_ERROR from Send_Data, Confirm – DEALLOCATE_ABEND from Send_Data, Confirm – PROG_ERROR from Send_Data, Confirm – RESOURCE_FAILURE from Send_Data, Confirm – OK from Send_Data, Confirm

	States	
	RESET	**SEND CONFIRM**
Inputs	**01**	**02**
SEND BUFFER	2a	/
END DMU	2b	/
ALLOCATION ERROR	/	1c
DEALLOCATE ABEND	/	1c
PROG ERROR	/	1d
RESOURCE FAILURE	/	1c
OK	/	1e

Output Code	**Function**
a	Signal LU 6.2 presentation services with SEND_DATA to send the encoded information to the partner DSU.
b	Signal LU 6.2 presentation services with CONFIRM to synchronize the processing between DS_Send and DS_Receive, following the sending of the DMU.
c	Signal FSM_DIST_ENCODE_CONTROL with CONVERSATION_FAILURE to indicate that some exception has occurred on the conversation between DS_Send and DS_Receive.
d	Signal FSM_DIST_ENCODE_CONTROL with RCV_SIDE_EXCEPT to indicate that DS_Receive has signalled that an exception has occurred on the receive side.
e	Signal FSM_DIST_ENCODE_CONTROL with OK to indicate that the LU 6.2 request was completed successfully.

FSM_SEMU_ENCODE

The function of this machine is to indicate to DS_Receive that an exception has occurred and the remainder of the DMU is not to follow. This function is requested from FSM_SEND_MGR signalling ENCODE_SEMU. The exception information required for building the exception code is passed to this machine.

The DS protocol for this processing is as follows:

- Issue a Send_Error to LU 6.2 presentation services.
- Build a SEMU indicating a forward abort. The SEMU contains the exception information.
- Issue a Send_Data, containing the SEMU, to LU 6.2 presentation services.

Exceptions can occur in the encoding process, and in sending the SEMU on the LU 6.2 conversation. For a further description of these exceptions and the processing that results, refer to "FSM_SEND_MGR—SND_EXPT State" on page 289.

Function:	This finite-state machine describes the functional processing for exception processing in DS_Send when the exception occurred on the sending side. For a further description, see "FSM_SEMU_ENCODE" on page 304. This FSM gets control from one of the following: • Signals from higher-level DS_Send finite-state machines: – ENCODE_SEMU from FSM_SEND_MGR • Signals from LU 6.2 presentation services: – OK from Send_Error, Send_Data – ALLOCATION_ERROR from Send_Error, Send_Data – RESOURCE_FAILURE from Send_Error, Send_Data – PROG_ERROR from Send_Error, Send_Data – DEALLOCATE_ABEND from Send_Error, Send_Data

	States		
	RESET	**SEND_ERR PEND**	**SEND SEMU**
Inputs	**01**	**02**	**03**
ENCODE SEMU	2a	/	/
OK	/	3b	1c
ALLOCATION ERROR	/	1e	1e
DEALLOCATE ABEND	/	1e	1e
PROG ERROR	/	1d	1d
RESOURCE FAILURE	/	1e	1e

Output Code	**Function**
a	Signal LU 6.2 presentation services with SEND_ERROR to indicate to the partner that an exception has been encountered on the sending side.
b	Build the SEMU and signal LU 6.2 presentation services with SEND_DATA to send the encoded information to the partner DSU.
c	Signal FSM_SEND_MGR with ERP_COMPLETE to indicate that exception processing has been completed and that appropriate deallocation should take place.
d	Signal FSM_SEND_MGR with RCV_SIDE_EXCEPT to indicate that while performing exception processing for an exception encountered on the sending side, the receiving side has indicated an exception.
e	Signal FSM_SEND_MGR with CONVERSATION_FAILURE to indicate that an exception occurred on the conversation while the exception information was being sent to DS_Receive.

FSM_REMU_DECODE

The function of this machine is to receive exception information from DS_Receive after DS_Receive has encountered an exception in receiving or parsing a DMU. This function is requested from FSM_SEND_MGR signalling DECODE_REMU.

The DS protocol for this exception is as follows:

- Issue Receive_And_Wait for the REMU.
- Parse the REMU containing the exception information.
- Issue Receive_And_Wait for the indication of Deallocate NORMAL.

Once FSM_REMU_DECODE receives and parses the REMU, it passes the REMU exception fields to FSM_SEND_MGR with the signal ERP_COMPLETE.

If an exception occurs in parsing the REMU, this machine issues Receive_And_Wait verbs until enough data has been received to comprise a complete REMU. If the next Receive_And_Wait verb does not return a Deallocate NORMAL as the return code, then it is treated as an abnormal case and logged as an improper DS usage of LU 6.2. Exceptions can occur while the MU is being received on the LU 6.2 conversation or in parsing the REMU. For a further description of these exceptions and the processing that results, see "FSM_SEND_MGR—RCV_EXPT State" on page 290.

Function:	This finite-state machine describes the functional processing for exception processing in DS_Send when the exception occurred on the receiving side. For a further description, see "FSM_REMU_DECODE" on page 306. This FSM gets control from one of the following: • Signals from higher-level DS_Send finite-state machines: – DECODE_REMU from FSM_SEND_MGR • Signals from PARSER – PARSE_NOT_OK—on PARSE_NOT_OK, Receive_And_Wait is issued until enough data has been received to comprise a complete REMU. – PARSE_COMPLETE • Signals from LU 6.2 presentation services: – RCV_AND_WAIT_SEND – RCV_AND_WAIT_CONFIRM – RCV_AND_WAIT_CONFIRM_DEALLOCATE – RCV_AND_WAIT_DATA_COMPLETE – RCV_AND_WAIT_DATA_INCOMPLETE – RCV_AND_WAIT_LL_TRUNCATED – RESOURCE_FAILURE – PROG_ERROR – DEALLOCATE_ABEND – DEALLOCATE_NORMAL

	States				
	RESET	RCV	PARSE	REMU EXPT	DEALLOC PEND
Inputs	01	02	03	04	05
DECODE REMU	2a	/	/	/	/
RCV AND WAIT DATA COMPLETE	/	3b	/	-a	1c
RCV AND WAIT DATA INCOMPLETE	/	3b	/	-a	1c
RCV AND WAIT LL TRUNCATED	/	-a	/	-a	1c
RCV AND WAIT SEND	/	1c	/	1c	1c
RCV AND WAIT CONFIRM	/	1c	/	1c	1c
RCV AND WAIT CONFIRM DEALLOCATE	/	1c	/	1c	1c
DEALLOCATE NORMAL	/	1d	/	1e	1f
DEALLOCATE ABEND	/	1d	/	1d	1d
PROG ERROR	/	1c	/	1c	1c
RESOURCE FAILURE	/	1d	/	1d	1d
PARSE NOT OK	/	/	4a	/	/
PARSE COMPLETE	/	/	5a	/	/

Output Code	Function
a	Signal LU 6.2 presentation services with RECEIVE_AND_WAIT to receive exception information from DS_Receive.
b	Signal PARSER with PARSE_REMU to parse the REMU.
c	Signal FSM_SEND_MGR with PROTOCOL_ERROR to indicate that the proper exception protocols were not followed for exception processing.
d	Signal FSM_SEND_MGR with CONVERSATION_FAILURE to indicate that exception processing was not successfully completed.
e	Signal FSM_SEND_MGR with SEND_SIDE_EXCEPT to indicate that an exception occurred in parsing the REMU.
f	Signal FSM_SEND_MGR with ERP_COMPLETE to indicate that exception processing has been completed, and that appropriate deallocation should take place.

DS Receive FSMs

DS Receive Overview

DS_Receive is shown here as a complex of machines whose function it is to receive DMUs on an LU 6.2 conversation from an instance of DS_Send in an adjacent DSU. These machines execute all the FS1 DS protocols for receiving DMUs, and all the protocols for both sender-and receiver-detected exceptions. There are several points to note about these protocols.

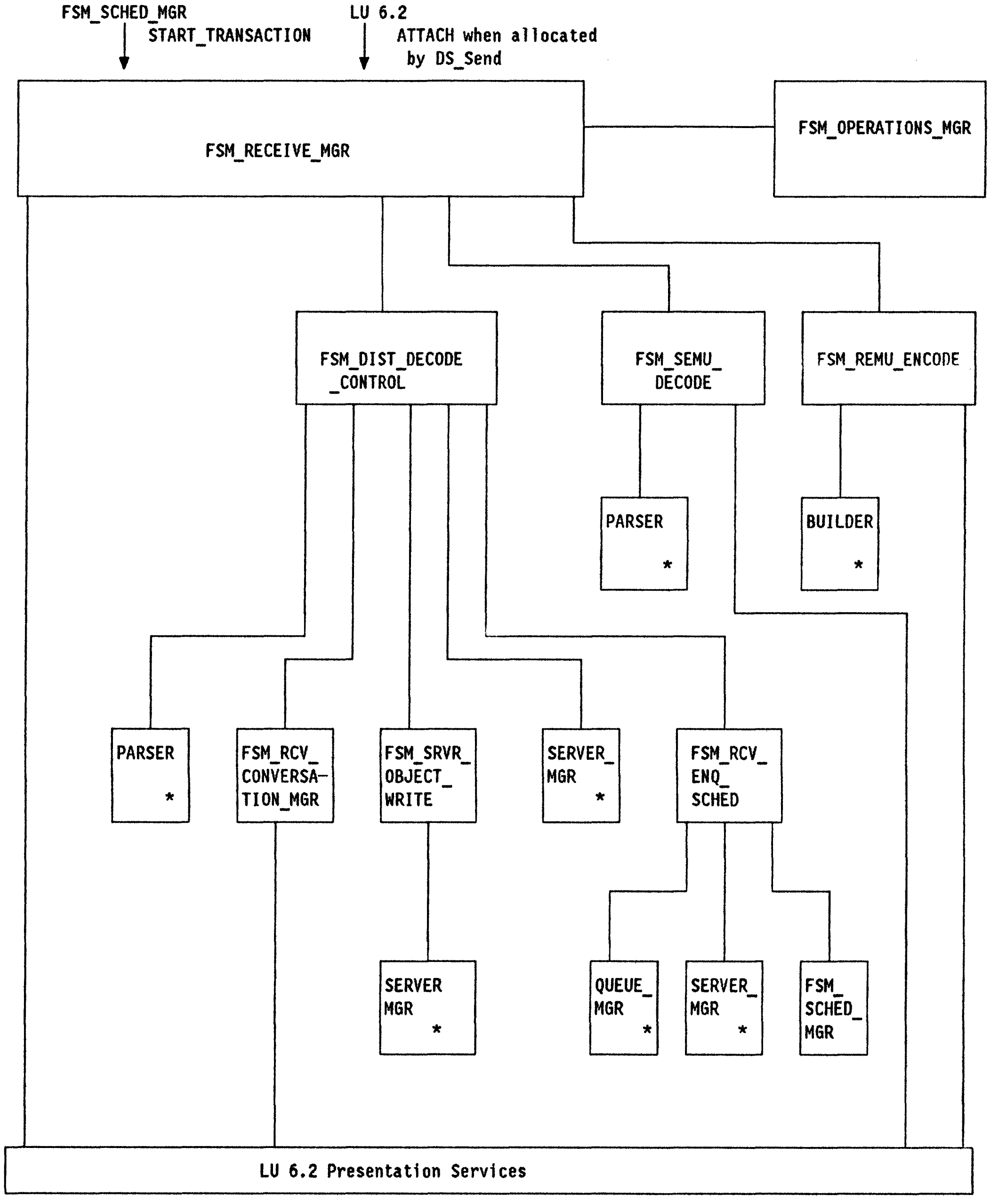
- Deallocate *type*(SYNC_LEVEL) is not issued by the formal model when sending, but all receivers can properly handle *what_received*(CONFIRM) and *what_received*(CONFIRM_DEALLOCATE) as parameters returned on the Receive_And_Wait verb.
- The number of Receive_And_Wait verbs issued in order to receive a DS MU is an implementation optimization choice. It will depend on the buffer sizes at the nodes at which the DSU resides, and on whether *fill*(BUFFER) or *fill*(LL) is used by the implementation.
- Each machine that issues verbs to LU 6.2 PS is listed below, with a summary of its function. All DS implementations follow the protocols embodied in these machines.
  - FSM_RECEIVE_MGR—Controls allocation and deallocation of the conversation resource for DS_Receive. It handles an Attach from DS_Send as well as sending an Attach to DS_Send. Some implementations may not send an Attach to DS_Send. It also issues the proper deallocation type for all situations, both exception and normal.
  - FSM_RCV_CONVERSATION_MGR—Controls the normal receiving of data and the Confirmed reply. It reports exception conditions to higher-level FSMs for proper exception-handling protocols and conversation deallocation.
  - FSM_SEMU_DECODE—Controls the protocols for exceptions detected by the remote conversation partner, (DS_Send) and handles any further exceptions that may occur during this processing.
  - FSM_REMU_ENCODE—Controls the sending, via LU 6.2 PS, of the REMU by the local conversation partner, and handles any further exceptions that may occur during this processing.

Although the structure of these machines may be peculiar to the formal model, all DS implementations generate these protocols.

Program Structure

These machines are structured so that a manager machine (FSM_RECEIVE_MGR) both allocates and deallocates the conversation. Once the conversation is started, machines lower in the hierarchy sequence the functions necessary to receive a distribution and parse the DMU. The manager machine receives reports on the progress of the transmission. If exceptions are detected, the manager machine causes one of two exception protocol machines to start. The exception protocol machines report on the progress of the exception information being transferred. If an exception occurs during an exception protocol, the manager machine takes appropriate action.

As each piece of the DMU is decoded, FSM_RCV_CONVERSATION_MGR is signalled to issue the appropriate LU 6.2 verb to retrieve the data from the LU. The manager machine and the exception protocol machines issue their own LU 6.2 verbs. The conversation state is kept in LU 6.2 PS. The state of the protocol is kept by a combination of machines and states in DS.



* indicates those finite-state machines not formally specified.

For more details regarding the finite-state machines, see:

- "FSM_RECEIVE_MGR" on page 313
- "FSM_SRVR_OBJECT_WRITE" on page 328
- "FSM_OPERATIONS_MGR" on page 342
- "FSM_RCV_CONVERSATION_MGR" on page 330
- "FSM_DIST_DECODE_CONTROL" on page 320
- "FSM_RCV_ENQ_SCHED" on page 324
- "FSM_SEMU_DECODE" on page 334
- "FSM_SCHED_MGR" on page 351
- "FSM_REMU_ENCODE" on page 338
- LU 6.2 presentation services
- "PARSER" on page 359
- "SERVER_MGR" on page 354
- "BUILDER" on page 359
- "QUEUE_MGR" on page 357

Figure 49. Ds_Receive FSM Hierarchy

FSM_RECEIVE_MGR

Sequences the functions required to receive and accept responsibility for a DMU sent from DS_Send over an LU 6.2 conversation to an instance of DS_Receive at another DSU. The request for this function can be from the local operator, or from DS_Send issuing an Allocate for DS_Receive.

The functions controlled by FSM_RECEIVE_MGR are:

- Allocation of the conversation to DS_Send, if started locally.
- Decoding the DMU format of the distribution and scheduling further processing.
- Deallocating the conversation when DS_Send signals that there are no more DMUs to flow on the conversation.
- Handling exception situations.

Processing stops when:

- An exception occurs while the conversation is being allocated.
- An exception occurs during the decoding of the DMU, or during the accessing of the server object or scheduling of further processing.
- An exception occurs during the accessing of the queues.
- An exception occurs on the conversation.

For exceptions detected by DS_Receive, FSM_RECEIVE_MGR signals a machine to handle the building and sending of the REMU to DS_Send. For exceptions detected by DS_Send, and reported to DS_Receive on the conversation, FSM_RECEIVE_MGR signals a machine to handle the decoding of the SEMU that contains exception information from DS_Send. For all exceptions, FSM_RECEIVE_MGR issues the correct form of the Deallocate verb to terminate the conversation, and signals FSM_OPERATIONS_MGR to log the exception and to notify the operator.

FSM_RECEIVE_MGR has 3 key states that determine DS actions in exception situations. In what follows, "local conversation partner" refers to the DSU receiving the DMU on the conversation, and "remote conversation partner" refers to the DSU sending the DMU on the conversation.

FSM_RECEIVE_MGR—DEC_PEND State: In this state, the receive manager waits for other machines to report on whether all the DMUs were received or whether an exception occurred while a DMU was being received. The signals in this state have the following meanings:

- SEND_SIDE_EXCEPT—LU 6.2 returned a PROG_ERROR_TRUNC or PROG_ERROR_NO_TRUNC on a verb issued by the local conversation partner, indicating that the remote conversation partner has detected an exception. FSM_RECEIVE_MGR signals another machine, FSM_SEMU_DECODE, to handle the exception protocol, including receiving the SEMU from DS_Send.
- RCV_SIDE_EXCEPT—Another machine in the hierarchy detected an exception while parsing the DMU, storing the server object or enqueuing the distribution. This is not an exception on the conversation, but rather an

exception detected at the local DSU. FSM_RECEIVE_MGR signals another machine, FSM_REMU_ENCODE, to handle the exception protocol, which includes sending a REMU to the remote conversation partner over the conversation.

- PROTOCOL_ERROR—If detected before the DMU is received completely, then LU 6.2 PS returned either *what_received*(SEND) or *what_received*(CONFIRM) to a Receive_And_Wait verb. If detected after the DMU has been received completely, then LU 6.2 PS returned either *what_received*(SEND) or *what_received*(DATA) to a Receive_And_Wait verb. The local conversation partner must assume that reliable communication cannot continue with the remote conversation partner. FSM_RECEIVE_MGR issues a Deallocate *type*(ABEND_PROG) to LU 6.2 PS to terminate the conversation abnormally.

- CONVERSATION_FAILURE—LU 6.2 PS returns one of the following return codes on a verb that DS has issued:

  - ALLOCATION_ERROR—A conversation could not be allocated, no data has been able to flow.

  - DEALLOCATE_NORMAL—Although this is an improper DS usage of LU 6.2, the conversation is no longer available, and the condition is signalled as a CONVERSATION_FAILURE. It is logged at the local DSU as an improper DS usage of LU 6.2, however.

  - Deallocate ABEND_PROG—The remote conversation partner has detected an exception that prevents further communication and has issued a Deallocate *type*(ABEND_PROG), or the remote LU has signalled a remote program abend condition.

  - RESOURCE_FAILURE—The session carrying the conversation has been lost.

  In each of these cases, the conversation is no longer active, and DS issues a Deallocate *type*(LOCAL) to terminate the local side of the conversation.

FSM_RECEIVE_MGR—SND_EXPT State: In this state, the receive manager waits for a report on the progress of the exception protocol following the detection of an exception by the remote conversation partner. The signals in this state have the following meanings:

- RCV_SIDE_EXCEPT— FSM_SEMU_DECODE detected an exception while parsing the SEMU. This is not an exception on the conversation, but rather an exception detected at the local DSU. FSM_SEMU_DECODE does not return this signal until a Deallocate NORMAL is returned by LU 6.2 PS to a Receive_And_Wait verb. FSM_RECEIVE_MGR completes the protocol by issuing a Deallocate *type*(LOCAL) verb.

- PROTOCOL_ERROR—If detected before the SEMU is received completely, then LU 6.2 PS returned either *what_received*(SEND) or *what_received*(CONFIRM) to a Receive_And_Wait verb. If detected after the SEMU has been received completely, then LU 6.2 PS returned *what_received*(SEND), *what_received*(CONFIRM), or *what_received*(DATA) to a Receive_And_Wait verb. This exception can also occur if LU 6.2 PS returns a PROG_ERROR_PURGING, PROG_ERROR_TRUNC, or PROG_ERROR_NO_TRUNC to a Receive_And_Wait verb. The local conversation partner must assume that reliable communication cannot continue with the remote conversation partner. FSM_RECEIVE_MGR issues a Deallocate *type*(ABEND_PROG) to terminate the conversation abnormally.

- CONVERSATION_FAILURE—LU 6.2 PS returns one of the following return codes on a verb that DS has issued:

  - Deallocate NORMAL—(if received before the SEMU is received completely). Although this is an improper DS usage of LU 6.2, the conversation is no longer available, and the condition is signalled as a CONVERSATION_FAILURE. It is logged at the local DSU as an improper DS usage of LU 6.2, however.

  - Deallocate ABEND_PROG—The remote conversation partner has detected an exception that prevents further communication and has issued a Deallocate *type*(ABEND_PROG), or the remote LU has signalled a remote program abend condition.

  - RESOURCE_FAILURE—The session carrying the conversation has been lost.

  In each of these cases, the conversation is no longer active, and DS issues a Deallocate *type*(LOCAL) to terminate the local side of the conversation.

- ERP_COMPLETE—A SEMU has been received successfully, and the remote conversation partner has terminated the conversation, which is indicated by a Deallocate NORMAL being returned by LU 6.2 PS to a Receive_And_Wait verb. FSM_RECEIVE_MGR terminates the local side of the conversation with Deallocate *type*(LOCAL) and logs the exception information sent by the remote conversation partner.

FSM_RECEIVE_MGR—RCV_EXPT State: In this state, the receive manager waits for a report on the progress of the exception protocol following the detection of an exception by the local conversation partner. The signals in this state have the following meanings:

- RCV_SIDE_EXCEPT— FSM_REMU_ENCODE detected an exception in building the REMU. This is not an exception on the conversation, but rather an exception detected at the local DSU. The exception protocol cannot continue and FSM_RECEIVE_MGR issues a Deallocate *type*(ABEND_PROG) to signal this to the remote conversation partner.

- PROTOCOL_ERROR—This exception occurs if LU 6.2 PS returns a PROG_ERROR_PURGING to a Send_Data verb used by FSM_REMU_ENCODE. The local conversation partner must assume that reliable communication cannot continue with the remote conversation partner. FSM_RECEIVE_MGR issues a Deallocate *type*(ABEND_PROG) to terminate the conversation abnormally.

- CONVERSATION_FAILURE—LU 6.2 PS returns one of the following return codes on a verb that DS has issued:

  - Deallocate ABEND_PROG—The remote conversation partner has detected an exception that prevents further communication, and has issued a Deallocate *type*(ABEND_PROG), or the remote LU has signalled a remote program abend condition.

  - RESOURCE_FAILURE—The session carrying the conversation has been lost.

  - Deallocate NORMAL—This return code is returned as a conversation exception in the following circumstances:

    The local and remote conversation partners have both issued Send_Error verbs, but the exception indication is not received by the remote conversation partner until it (the remote conversation partner) issues a Deallocate *type*(FLUSH). In this case, the conversation is gone, but no improper DS usage of LU 6.2 has occurred. The local conversation partner cannot send the REMU, and must simply log the exception locally.

  In each of the CONVERSATION_FAILURE cases, the conversation is no longer active, and DS issues a Deallocate *type*(LOCAL) to terminate the local side of the exception.

- ERP_COMPLETE—A REMU has been sent successfully to the remote conversation partner. DS issues a Deallocate *type*(FLUSH) to terminate the conversation and signal the remote conversation partner that the exception protocol is complete.

Function: This finite-state machine describes the functional processing for the receive manager of DS_Receive transaction program. For a further description, see "FSM_RECEIVE_MGR" on page 313.

This FSM gets control from one of the following:

- Signals from finite-state machines providing common services:
  - START_TRANSACTION from FSM_SCHED_MGR
  - OPERATIONS_COMPLETE from FSM_OPERATIONS_MGR
- Signals from LU 6.2 presentation services:
  - ATTACH from LU 6.2 when DS_Send issues Allocate
  - OK from Allocate, Deallocate FLUSH, Deallocate LOCAL, Deallocate ABEND
  - ALLOC_PARAM_ERROR from Allocate
  - ALLOCATION_ERROR from Allocate
- Signals from lower-level DS_Receive finite-state machines:
  - END_DMUS from FSM_DIST_DECODE_CONTROL
  - SEND_SIDE_EXCEPT from FSM_DIST_DECODE_CONTROL
  - RCV_SIDE_EXCEPT
    - from FSM_DIST_DECODE_CONTROL
    - from FSM_REMU_ENCODE
    - from FSM_SEMU_DECODE
  - PROTOCOL_ERROR
    - from FSM_DIST_DECODE_CONTROL
    - from FSM_REMU_ENCODE
    - from FSM_SEMU_DECODE
  - CONVERSATION_FAILURE
    - from FSM_DIST_DECODE_CONTROL
    - from FSM_REMU_ENCODE
    - from FSM_SEMU_DECODE
  - ERP_COMPLETE from FSM_SEMU_DECODE, FSM_REMU_ENCODE

	States							
	RESET	ALLOC	DEC PEND	SND EXPT	RCV EXPT	ERP	OPS PEND	END
Inputs	01	02	03	04	05	06	07	08
ATTACH	3i	/	/	/	/	/	/	/
START TRANSACTION	2a	/	/	/	/	/	/	/
OK	/	3b	/	/	/	7c	/	1
ALLOC PARAM ERROR	/	7c	/	/	/	/	/	/
ALLOCATION ERROR	/	6d	/	/	/	/	/	/
END DMUS	/	/	8d	/	/	/	/	/
SEND SIDE EXCEPTION	/	/	4e	/	/	/	/	/.
RCV SIDE EXCEPTION	/	/	5f	6d	6g	/	/	/
PROTOCOL ERROR	/	/	6g	6g	6g	/	/	/
CONVERSATION FAILURE	/	/	6d	6d	6d	/	/	/
ERP COMPLETE	/	/	/	6d	6h	/	/	/
OPERATIONS COMPLETE	/	/	/	/	/	/	1	/

Output Code	Function
a	Signal LU 6.2 presentation services with ALLOCATE to establish the conversation with DS_Send.
b	Signal FSM_DIST_DECODE_CONTROL with DECODE to control the receiving and decoding of the DMU from DS_Send.
c	Signal FSM_OPERATIONS_MGR with LOG_MESSAGE_EXCEPT for logging and operator messages.
d	Signal LU 6.2 presentation services with DEALLOCATE_LOCAL to deallocate the conversation locally.
e	Signal FSM_SEMU_DECODE with DECODE_SEMU to control the protocols for an exception encountered by DS_Send.
f	Signal FSM_REMU_ENCODE with ENCODE_REMU to control the building and sending of the REMU used to provide exception information from DS_Receive to DS_Send.
g	Signal LU 6.2 presentation services with DEALLOCATE_ABEND to deallocate the conversation abnormally.
h	Signal LU 6.2 presentation services with DEALLOCATE_FLUSH to signal the partner that no more DMUs should be sent.
i	Signal LU 6.2 presentation services with GET_ATTRIBUTES. Signal FSM_DIST_DECODE_CONTROL with DECODE to control the receiving and decoding of the DMU from DS_Send.

FSM_DIST_DECODE_CONTROL

This machine sequences the receiving of the DMU at a DSU and responds properly to the Confirm request from the remote conversation partner.

This machine checks that the parts of the DMU are received and that they can be parsed without exceptions. Once the entire DMU has been received and parsed, the machine waits for a Confirm from the remote conversation partner (DS_Send) and causes the distribution to be put on the router-director queue. After these operations have been successfully completed, Confirmed is signalled to the remote conversation partner.

Exceptions in decoding result in a RCV_SIDE_EXCEPT signal to FSM_RECEIVE_MGR. Exceptions detected by DS_Send indicating that DS_Receive has violated the DS protocols regarding usage of LU 6.2 result in a PROTOCOL_ERROR signal to FSM_RECEIVE_MGR. Any other exceptions detected by DS_Send during transmission of the DMU result in a SEND_SIDE_EXCEPT signal to FSM_RECEIVE_MGR, and exceptions on the conversation result in a CONVERSATION_FAILURE signal to FSM_RECEIVE_MGR.

Exceptions in receiving a DMU are reported by FSM_RCV_CONVERSATION_MGR. If the initial call to FSM_RCV_CONVERSATION_MGR does not return a signal of OK, this machine passes the available exception information to FSM_RECEIVE_MGR. If a subsequent call to FSM_RCV_CONVERSATION_MGR results in an exception, this machine signals FSM_SRVR_OBJECT_WRITE with CLEANUP_OBJECT to delete any portion of the server object that may exist.

When a DMU/Confirm/Confirmed sequence is complete, this machine enters a state to wait for the next DMU to flow on the LU 6.2 conversation. When the remote conversation partner deallocates the conversation normally, this machine signals FSM_RECEIVE_MGR with END_DMUS to indicate the conversation should be terminated locally.

If FSM_RCV_CONVERSATION_MGR returns a signal of DEALLOCATE_NORMAL before the DMU/Confirm/Confirmed sequence is complete, this machine signals FSM_RECEIVE_MGR with CONVERSATION_FAILURE.

The exceptions that can be signalled from this machine are SEND_SIDE_EXCEPT, RCV_SIDE_EXCEPT, CONVERSATION_FAILURE, and PROTOCOL_ERROR. The handling of these exceptions is described in "FSM_RECEIVE_MGR—DEC_PEND State" on page 313. Although it is not shown in this machine, for an unsuccessful server function, FSM_OPERATIONS_MGR should also be signalled with MESSAGE_TO_OPERATOR and passed the message that the DS lock count on the server object could not be successfully decremented.

Function: This finite-state machine describes the functional processing for controlling the receiving and parsing of the DMUs. For a further description, see "FSM_DIST_DECODE_CONTROL" on page 320.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines:
  - DECODE from FSM_RECEIVE_MGR
- Signals from lower-level DS_Receive finite-state machines:
  - OBJECT_OK from FSM_SRVR_OBJECT_WRITE
  - OBJECT_NOT_OK from FSM_SRVR_OBJECT_WRITE
  - OK from FSM_RCV_CONVERSATION_MGR
  - SEND_SIDE_EXCEPT from FSM_RCV_CONVERSATION_MGR
  - CONVERSATION_FAILURE from FSM_RCV_CONVERSATION_MGR
  - PROTOCOL_ERROR from FSM_RCV_CONVERSATION_MGR
  - END_DMUS from FSM_RCV_CONVERSATION_MGR
  - ENQ_SCHED_OK from FSM_RCV_ENQ_SCHED
  - ENQ_SCHED_NOT_OK from FSM_RCV_ENQ_SCHED
- Signals from finite-state machines providing common services:
  - OBJECT_OK from SERVER_MGR
  - OBJECT_NOT_OK from SERVER_MGR
- Signals from "PARSER" on page 359
  - PARSE_OK
  - PARSE_OK_OBJECT
  - PARSE_COMPLETE
  - PARSE_NOT_OK

	States												
	RST	RCV FIRST	RCV	PARS	OBJ PEND	END DMU	CON-FIRM PEND	ENQ SCHD	CON-FMED	SEND ERP	CONV ERP	PROT ERP	RCV ERP
Inputs	01	02	03	04	05	06	07	08	09	10	11	12	13
DECODE	2a	/	/	/	/	/	/	/	/	/	/	/	/
OK	/	4b	4b	/	/	/	8f	/	2a	/	/	/	/
SEND SIDE EXCEPTION	/	1h	10m	/	/	/	10o	/	/	/	/	/	/
CONVERSATION FAILURE	/	1i	11m	/	/	/	11o	/	/	/	/	/	/
PROTOCOL ERROR	/	1j	12m	/	/	/	12o	/	/	/	/	/	/
END DMUS	/	1k	11m	/	/	/	/	/	/	/	/	/	/
PARSE OK	/	/	/	3a	/	/	/	/	/	/	/	/	/
PARSE OK OBJECT	/	/	/	5c	/	/	/	/	/	/	/	/	/
PARSE COMPLETE	/	/	/	6d	/	/	/	/	/	/	/	/	/
PARSE NOT OK	/	/	/	13m	/	/	/	/	/	/	/	/	/
OBJECT OK	/	/	/	/	3a	7e	/	/	/	1h	1i	1j	1n
OBJECT NOT OK	/	/	/	/	1n	1n	/	/	/	1h	1i	1j	1n
ENQ SCHED OK	/	/	/	/	/	/	/	9g	/	/	/	/	/
ENQ SCHED NOT OK	/	/	/	/	/	/	/	1n	/	/	/	/	/

Output Code	Function
a	Signal FSM_RCV_CONVERSATION_MGR with RECEIVE_BUFFER to receive information from LU 6.2 PS.
b	Signal PARSER with PARSE to parse information received.
c	Signal FSM_SRVR_OBJECT_WRITE with WRITE_OBJECT to write the information and initialize for writing if not yet initialized.
d	Signal FSM_SRVR_OBJECT_WRITE with END_DMU to terminate writing.
e	Signal FSM_RCV_CONVERSATION_MGR with WAIT_CONFIRM to indicate that a DMU has been completely received and that the CONFIRM indicator should be the next indicator from DS_Send.
f	Signal FSM_RCV_ENQ_SCHED with ENQ_SCHED to place the distribution on the router-director queue and schedule DS_Router_Director.
g	Signal FSM_RCV_CONVERSATION_MGR with CONFIRMED to indicate that DS_Receive has accepted responsibility for the DMU and that Confirmed should be sent to DS_Send.
h	Signal FSM_RECEIVE_MGR with SEND_SIDE_EXCEPT to indicate that an exception has occurred in DS_Send and that the sender exception protocols should be followed.
i	Signal FSM_RECEIVE_MGR with CONVERSATION_FAILURE to indicate that an exception has occurred in the conversation between DS_Send and DS_Receive.
j	Signal FSM_RECEIVE_MGR with PROTOCOL_ERROR to indicate that a protocol exception has occurred in the conversation between DS_Send and DS_Receive.
k	Signal FSM_RECEIVE_MGR with END_DMUS to indicate that the Deallocate NORMAL return code has been received from DS_Send indicating that DS_Send will send no more DMUs.
m	Signal FSM_SRVR_OBJECT_WRITE with CLEANUP_OBJECT to delete any portion of the object that may exist. This is for exception cleanup and consists of a Terminate_Write, if necessary, followed by decrementing the DS lock count.
n	Signal FSM_RECEIVE_MGR with RCV_SIDE_EXCEPT to indicate that an exception has occurred in parsing or in scheduling further processing, and that the receiver exception protocols should be followed.
o	Signal SERVER_MGR with DECREMENT_OBJ_LOCK to indicate that the DS lock count should be decremented and the object deleted if the lock count reaches 0. If no server object exists, SERVER_MGR returns OBJECT_OK.

FSM_RCV_ENQ_SCHED

This machine controls the placing of the distribution on the router-director queue and the scheduling of DS_Router_Director. The request for this function is from FSM_DIST_DECODE_CONTROL.

The sequence of functions performed is as follows:

- Issue WRITEQ to put the distribution on the router-director queue.
- Issue the request to start the DS_Router_Director transaction program.
- Issue RELEASEQ to remove the in-use mark from the entry on the router-director queue so that it is available for processing.

Exceptions can occur in enqueuing or scheduling. For enqueuing or scheduling exceptions, this machine signals to SERVER_MGR to decrement the server object DS lock count, and delete the server object if both the DS and agent lock counts are 0. For scheduling exceptions, the queue entry is dequeued.

For any exception, this machine signals ENQ_SCHED_NOT_OK to FSM_DIST_DECODE_CONTROL. Although it is not shown in this machine, for an unsuccessful server, queue, or scheduling function, FSM_OPERATIONS_MGR should also be signalled with MESSAGE_TO_OPERATOR and passed the message that the function could not be completed, and that, while FSM_RCV_ENQ_SCHED was handling the exception, another exception occurred.

Function: This finite-state machine describes the functional processing in DS_Receive to place the distribution on the router-director queue and schedule DS_Router_Director. For a further description, see "FSM_RCV_ENQ_SCHED" on page 324.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines:
  - ENQ_SCHED from FSM_DIST_DECODE_CONTROL
- Signals from finite-state machines providing common services:
  - OBJECT_OK from SERVER_MGR
  - OBJECT_NOT_OK from SERVER_MGR
  - QUEUE_OK from QUEUE_MGR
  - QUEUE_NOT_OK from QUEUE_MGR
  - SCHED_FUNCTION_OK from FSM_SCHED_MGR
  - SCHED_FUNCTION_NOT_OK from FSM_SCHED_MGR

	States					
	RESET	ENQ PEND	SCHED PEND	RELQ PEND	ERP REL	ERP DEQ
Inputs	01	02	03	04	05	06
ENQ SCHED	2a	/	/	/	/	/
QUEUE OK	/	3b	/	1d	/	5c
QUEUE NOT OK	/	5c	/	5c	/	5c
SCHED FUNCTION OK	/	/	4g	/	/	/
SCHED FUNCTION NOT OK	/	/	6e	/	/	/
OBJECT OK	/	/	/	/	1f	/
OBJECT NOT OK	/	/	/	/	1f	/

Output Code	Function
a	Signal QUEUE_MGR with WRITEQ to place the distribution on the router-director queue.
b	Signal FSM_SCHED_MGR with START_REQUEST to schedule DS_Router_Director.
c	Signal SERVER_MGR with DECREMENT_OBJ_LOCK to indicate that the DS lock count should be decremented and the object deleted if both the DS and agent lock counts reach 0. If no server object exists, SERVER_MGR returns OBJECT_OK.
d	Signal FSM_DIST_DECODE_CONTROL with ENQ_SCHED_OK to indicate that the distribution was successfully placed on the router-director queue and that DS_Router_Director was scheduled.
e	Signal QUEUE_MGR with DEQ to remove the distribution from the router-director queue.
f	Signal FSM_DIST_DECODE_CONTROL with ENQ_SCHED_NOT_OK to indicate that an exception occurred in placing the distribution on the router-director queue, or in scheduling DS_Router_Director. Cleanup of the queues and the object has taken place.
g	Signal QUEUE_MGR with RELEASEQ to remove the in-use mark from the entry on the router-director queue. The entry is then available for processing by DS_Router_Director.

FSM_SRVR_OBJECT_WRITE

This machine controls the writing of the server object. The request for this function is from FSM_DIST_DECODE_CONTROL.

For an input signal of WRITE_OBJECT, this machine signals WRITE to the server manager. For an input signal of END_DMU, this machine signals TERMINATE_WRITE to the server manager. For an input signal of CLEANUP_OBJECT, prior to a successful Initiate_Write operation, this machine signals OBJECT_OK. For any subsequent signal of CLEANUP_OBJECT, this machine signals CLEANUP_OBJECT to SERVER_MGR to perform a Terminate_Write operation and decrement the DS server object lock.

The general server is used to write the server object. If direct store is to be performed, the receive-time processing required for direct store is performed prior to writing the server object. Based on the receive-time processing, the specific server can be used to store the server object rather than the general server. For a description of direct store, see Chapter 1. The server name is passed to SERVER_MGR.

Exceptions can occur in writing the server object. For any exception occurring after the initial WRITE, a Terminate_Write is issued and the DS lock count decremented. In case of a failure from the server, FSM_OPERATIONS_MGR is signalled with MESSAGE_TO_OPERATOR to notify the operator that a server object cannot be accessed as requested. This function is not shown in this machine.

Exceptions in writing result in an OBJECT_NOT_OK signal to FSM_DIST_DECODE_CONTROL and RCV_SIDE_EXCEPT signal to FSM_RECEIVE_MGR. For a further description of these exceptions and the processing that results, see "FSM_RECEIVE_MGR—DEC_PEND State" on page 313.

Function: This finite-state machine describes the functional processing for writing the object. For a further description, see "FSM_SRVR_OBJECT_WRITE" on page 328.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines:
  - WRITE_OBJECT from FSM_DIST_DECODE_CONTROL
  - END_DMU from FSM_DIST_DECODE_CONTROL
  - CLEANUP_OBJECT from FSM_DIST_DECODE_CONTROL
- Signals from machines providing common services:
  - OBJECT_OK from SERVER_MGR
  - OBJECT_NOT_OK from SERVER_MGR

	States				
	RESET	WRT	RCV PEND	TRM WRT	ERP
Inputs	01	02	03	04	05
WRITE OBJECT	2a	/	2a	/	/
END DMU	-b	/	4f	/	/
CLEANUP OBJECT	-b	/	5e	/	/
OBJECT OK	/	3b	/	1b	1d
OBJECT NOT OK	/	5e	/	5c	1d

Output Code	Function
a	Signal SERVER_MGR with WRITE to write a portion of the object.
b	Signal FSM_DIST_DECODE_CONTROL with OBJECT_OK to indicate that the portion of the object has been written, and the access terminated if the end of the DMU has been reached. If no object exists, OK is also signalled.
c	Signal SERVER_MGR with DECREMENT_OBJ_LOCK to decrement the DS lock count.
d	Signal FSM_DIST_DECODE_CONTROL with OBJECT_NOT_OK to indicate that an exception has occurred during the writing of the object.
e	Signal SERVER_MGR with CLEANUP_OBJECT to delete any portion of the object that may exist. This is for exception cleanup and consists of a Terminate_Write, if necessary, followed by decrementing the DS lock count.
f	Signal SERVER_MGR with TERMINATE_WRITE to terminate the access to the object.

FSM_RCV_CONVERSATION_MGR

FSM_RCV_CONVERSATION_MGR issues Receive_And_Wait for receiving the DMU and the confirm indicator, and issues Confirmed to indicate that DS_Receive accepts responsibility for the DMU. This machine receives input signals from FSM_DIST_DECODE_CONTROL.

Figure 50 classifies the possible exceptions.

Return Code	Reported As	Explanation
ALLOCATION ERROR	CONVERSATION_FAILURE	Conversation was never allocated.
DEALLOCATE NORMAL	CONVERSATION_FAILURE	In some cases, a protocol error was detected; conversation terminated.
DEALLOCATE ABEND PROG	CONVERSATION_FAILURE	Abend condition or protocol error was detected by partner TP.
PROG ERROR TRUNC	SEND_SIDE_ERROR	DS_Send issued a Send_Error indicating an error.
PROG ERROR NO TRUNC	SEND_SIDE_ERROR	DS_Send issued a Send_Error indicating an error.
RESOURCE FAILURE	CONVERSATION_FAILURE	Conversation was lost because the session was lost.

Figure 50. Classification of NOT-OK Return Codes by FSM_RCV_CONVERSATION_MGR

Function: This finite-state machine describes the functional processing to control receiving the data from an adjacent DSU. For a further description, see "FSM_RCV_CONVERSATION_MGR" on page 330.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines:
  - RECEIVE_BUFFER from FSM_DIST_DECODE_CONTROL
  - WAIT_CONFIRM from FSM_DIST_DECODE_CONTROL
  - CONFIRMED from FSM_DIST_DECODE_CONTROL
- Signals from LU 6.2 presentation services:
  - ALLOCATION_ERROR from Receive_And_Wait
  - RESOURCE_FAILURE from Receive_And_Wait
  - PROG_ERROR from Receive_And_Wait
  - DEALLOCATE_ABEND from Receive_And_Wait
  - DEALLOCATE_NORMAL from Receive_And_Wait
  - RECEIVE_AND_WAIT_DATA_COMPLETE from Receive_And_Wait
  - RECEIVE_AND_WAIT_DATA_INCOMPLETE from Receive_And_Wait
  - RECEIVE_AND_WAIT_LL_TRUNCATED from Receive_And_Wait
  - RECEIVE_AND_WAIT_SEND from Receive_And_Wait
  - RECEIVE_AND_WAIT_CONFIRM from Receive_And_Wait
  - RECEIVE_AND_WAIT_CONFIRM_DEALLOCATE from Receive_And_Wait

	States			
	RESET	RCV	CONFIRM PEND	DEALLOC
Inputs	01	02	03	04
RECEIVE BUFFER	2a	/	/	1f
WAIT CONFIRM	3a	/	/	/
CONFIRMED	-b	/	/	/
RCV AND WAIT DATA COMPLETE	/	1c	1e	/
RCV AND WAIT DATA INCOMPLETE	/	1c	1e	/
RCV AND WAIT LL TRUNCATED	/	-a	1e	/
RCV AND WAIT SEND	/	1e	1e	/
RCV AND WAIT CONFIRM	/	1e	1c	/
RCV AND WAIT CONFIRM DEALLOCATE	/	1e	4c	/
ALLOCATION ERROR	/	1d	/	/
DEALLOCATE NORMAL	/	1f	1d	/
DEALLOCATE ABEND	/	1d	1d	/
PROG ERROR	/	1g	1g	/
RESOURCE FAILURE	/	1d	1d	/

Output Code	Function
a	Signal LU 6.2 presentation services with RECEIVE_AND_WAIT to indicate that data can be accepted.
b	Signal LU 6.2 presentation services with CONFIRMED to indicate that responsibility for the DMU has been accepted by DS_Receive. Signal FSM_DIST_DECODE_CONTROL with OK to indicate that the LU 6.2 request completed successfully.
c	Signal FSM_DIST_DECODE_CONTROL with OK to indicate that the LU 6.2 request completed successfully.
d	Signal FSM_DIST_DECODE_CONTROL with CONVERSATION_FAILURE to indicate that an exception occurred on the conversation between DS_Send and DS_Receive.
e	Signal FSM_DIST_DECODE_CONTROL with PROTOCOL_ERROR to indicate that an error in the protocols has occurred between DS_Send and DS_Receive.
f	Signal FSM_DIST_DECODE_CONTROL with END_DMUS. This indicates that either a Deallocate NORMAL has been received from DS_Send, or the conversation is already deallocated; i.e., DS_Send issued Deallocate *TYPE*(SYNC_LEVEL) on the previous DMU. Receiving a Deallocate NORMAL after the *prefix* is a protocol error, but is treated as a conversation failure because the conversation resource is no longer available.
g	Signal FSM_DIST_DECODE_CONTROL with SEND_SIDE_EXCEPT to indicate that DS_Send has encountered an exception.

FSM_SEMU_DECODE

This machine receives exception information from DS_Send after DS_Send has encountered an exception in building or sending a DMU. This function is requested from FSM_RECEIVE_MGR signalling DECODE_SEMU.

The DS protocol for this exception is:

- Issue Receive_And_Wait for the SEMU forward abort indicator.
- Parse the SEMU containing the exception information.
- Issue Receive_And_Wait for the Deallocate NORMAL.

Once FSM_SEMU_DECODE receives and parses the SEMU, it passes the exception fields to FSM_RECEIVE_MGR with the signal ERP_COMPLETE. Exceptions can occur in receiving the SEMU on the LU 6.2 conversation or in parsing the exception information. For a further description of these exceptions and the processing that results, see "FSM_RECEIVE_MGR—SND_EXPT State" on page 314.

Function: This finite-state machine describes the functional processing for exception processing in DS_Receive when the exception occurred on the sending side. For a further description, see "FSM_SEMU_DECODE" on page 334.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines:
  - DECODE_SEMU from FSM_RECEIVE_MGR
- Signals from PARSER:
  - PARSE_NOT_OK
  - PARSE_COMPLETE
- Signals from LU 6.2 presentation services:
  - RCV_AND_WAIT_SEND
  - RCV_AND_WAIT_CONFIRM
  - RCV_AND_WAIT_CONFIRM_DEALLOCATE
  - RCV_AND_WAIT_DATA_COMPLETE
  - RCV_AND_WAIT_DATA_INCOMPLETE
  - RCV_AND_WAIT_LL_TRUNCATED
  - RESOURCE_FAILURE
  - PROG_ERROR
  - DEALLOCATE_ABEND
  - DEALLOCATE_NORMAL

	States				
	RESET	**RCV**	**DECODE SEMU**	**DEALLOC**	**RCV ERP**
Inputs	**01**	**02**	**03**	**04**	**05**
DECODE SEMU	2a	/	/	/	/
RCV AND WAIT DATA COMPLETE	/	3b	/	1c	1c
RCV AND WAIT DATA INCOMPLETE	/	-a	/	1c	1c
RCV AND WAIT LL TRUNCATED	/	-a	/	1c	1c
RCV AND WAIT SEND	/	1c	/	1c	1c
RCV AND WAIT CONFIRM	/	1c	/	1c	1c
RCV AND WAIT CONFIRM DEALLOCATE	/	1c	/	1c	1c
DEALLOCATE NORMAL	/	1e	/	1d	1f
DEALLOCATE ABEND	/	1e	/	1e	1e
PROG ERROR	/	1c	/	1c	1c
RESOURCE FAILURE	/	1e	/	1e	1e
PARSE NOT OK	/	/	5a	/	/
PARSE COMPLETE	/	/	4a	/	/

Output Code	Function
a	Signal LU 6.2 presentation services with RECEIVE_AND_WAIT to receive exception information from DS_Send.
b	Signal PARSER with PARSE_SEMU to parse the SEMU.
c	Signal FSM_RECEIVE_MGR with PROTOCOL_ERROR to indicate that the proper protocols were not followed for exception processing.
d	Signal FSM_RECEIVE_MGR with ERP_COMPLETE to indicate that exception processing completed and that appropriate deallocation should take place.
e	Signal FSM_RECEIVE_MGR with CONVERSATION_FAILURE to indicate that exception processing did not successfully complete.
f	Signal FSM_RECEIVE_MGR with RCV_SIDE_EXCEPT to indicate that an exception occurred in parsing the SEMU.

FSM_REMU_ENCODE

This machine indicates to DS_Send that an exception has occurred during the receiving or parsing of the DMU and passes to DS_Send the exception information in the REMU. The function is requested by FSM_RECEIVE_MGR signalling ENCODE_REMU. The exception information required for building the REMU is passed to FSM_REMU_ENCODE.

The DS protocol for this processing is:

- Issue a Send_Error to LU 6.2 presentation services.
- Build an REMU containing the exception information and the name of the DSU detecting the exception.
- Issue Send_Data(s) to LU 6.2 presentation services for the REMU.

Exceptions can occur in the encoding process, and in sending the REMU on the LU 6.2 conversation. For a further description of these exceptions and the processing that results, refer to "FSM_RECEIVE_MGR—RCV_EXPT State" on page 315.

Function: This finite-state machine describes the functional processing for exception processing in DS_Receive when the exception occurred on the receiving side. For a further description, see "FSM_REMU_ENCODE" on page 338.

This FSM gets control from one of the following:

- Signals from higher-level DS_Receive finite-state machines:
  - ENCODE_REMU from FSM_RECEIVE_MGR
- Signals from LU 6.2 presentation services:
  - OK from Send_Error, Send_Data
  - RESOURCE_FAILURE from Send_Error, Send_Data
  - PROG_ERROR from Send_Data
  - DEALLOCATE_ABEND from Send_Error, Send_Data
  - DEALLOCATE_NORMAL from Send_Error

	States		
	RESET	SEND_ERR PEND	SEND REMU
Inputs	01	02	03
ENCODE REMU	2a	/	/
OK	/	3b	1c
DEALLOCATE ABEND	/	1d	1d
PROG ERROR	/	/	1e
RESOURCE FAILURE	/	1d	1d
DEALLOCATE NORMAL	/	1d	/

Output Code	Function
a	Signal LU 6.2 presentation services with SEND_ERROR to indicate to the partner that an exception was encountered on the receiving side.
b	Build a REMU with the appropriate exception code (using a *prefix correlation* if one existed in the DMU *prefix*), and signal LU 6.2 presentation services with SEND_DATA to send the encoded information to the partner DSU.
c	Signal FSM_RECEIVE_MGR with ERP_COMPLETE to indicate that exception processing has been completed and that appropriate deallocation should take place.
d	Signal FSM_RECEIVE_MGR with CONVERSATION_FAILURE to indicate that an exception has occurred in the conversation between DS_Send and DS_Receive.
e	Signal FSM_RECEIVE_MGR with PROTOCOL_ERROR to indicate that the proper exception protocols were not followed for exception processing.

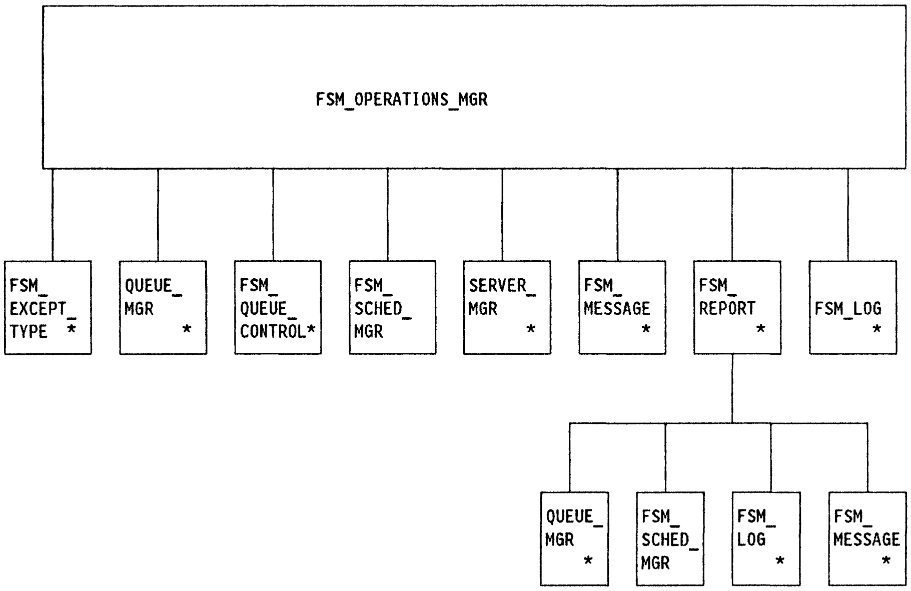
Common Services

These machines provide functions to all the other machines in the formal model. Some implementations will provide some of these functions as part of their runtime environment; others will package them differently. Some of these functions are provided by all DS implementations:

- FSM_OPERATIONS_MGR
  - All DS implementations support some form of the release and hold functions for the queues, if queues are supported at the DS level of implementation.
  - All DS implementations provide some logging when exceptions are detected. See Appendix C and Appendix E for rules for logging, and when logging takes place.
- FSM_SCHED_MGR

  The formal model presents two scheduling triggering mechanisms: Scheduling based on the time of day, and scheduling based on the number of next-DSU queue entries. These scheduling mechanisms are electives. See Appendix C for electives.

Operations



* indicates those finite-state machines not formally specfied.

For more details regarding the finite-state machines see:

- "FSM_OPERATIONS_MGR" on page 342
- "FSM_SCHED_MGR" on page 351
- "FSM_EXCEPT_TYPE" on page 348
- "QUEUE_MGR" on page 357
- "FSM_QUEUE_CONTROL" on page 348
- "SERVER_MGR" on page 354
- "FSM_MESSAGE" on page 348
- "FSM_REPORT" on page 349
- "FSM_LOG" on page 349

Figure 51. Operations FSM Hierarchy

FSM_OPERATIONS_MGR

This machine controls the logging of exceptions, messages to the operator, queue control, and generation of reports. FSM_OPERATIONS_MGR signals other FSMs to perform each of the functions. The requests for these functions are as follows:

1. Exceptions in DS_Receive

   FSM_RECEIVE_MGR signals FSM_OPERATIONS_MGR with LOG_MESSAGE_EXCEPT to perform the following:

   - Logging of an exception encountered in DS_Receive or indicated to DS_Receive by DS_Send.
   - Sending a message to the operator.

2. Exceptions in DS_Send

   FSM_SEND_MGR signals FSM_OPERATIONS_MGR with:

   a. SEND_MU_EXCEPT to perform the following:
      - Logging of an exception encountered in DS_Send or indicated to DS_Send by DS_Receive.
      - Sending a message to the operator.
      - Queue control.
      - Determination of retriable and nonretriable exceptions.
      - Releasing of the queue entry if retriable.
      - Removing of the queue entry, decrementing the server object DS lock count, and generating a report if nonretriable.

   b. LOG_MESSAGE_EXCEPT, to perform the following, if the exception is not on a particular MU:
      - Logging of the exception condition.
      - Sending a message to the operator.

3. Exceptions in DS_Router_Director

   FSM_ROUTING_DIRECTING_MGR signals this machine with LOG_MESSAGE_EXCEPT, to perform the following, if the exception is not on a particular MU:

   - Logging of the exception condition.
   - Sending a message to the operator.

   FSM_DIRECTING_MGR and FSM_ROUTING_MGR signal FSM_OPERATIONS_MGR with ROUTING_DIRECTING_EXCEPT to perform the following:

   - Logging of the exception condition.
   - Sending a message to the operator.
   - Generation of a report.

4. Exceptions during exception processing

   Any machine signals FSM_OPERATIONS_MGR with MESSAGE_TO_OPERATOR when exception processing is being performed, and an exception occurs during this processing. FSM_OPERATIONS_MGR signals FSM_MESSAGE to send a message to the operator. This generally signals conditions that the DSU cannot recover and that require assistance from the operator to clear up.

5. Other applications

   An operator signals FSM_OPERATIONS_MGR with SEND_RELEASE_HOLD, or RCV_RELEASE_HOLD to start DS_Send or DS_Receive transaction programs.

Function: This finite-state machine describes the functional processing in operations for generating reports, logging, sending operator messages, and handling queue control. For a further description, see "FSM_OPERATIONS_MGR" on page 342.

This FSM gets control from one of the following:

- Signals from routing and directing finite-state machines:
  - ROUTING_DIRECTING_EXCEPT from FSM_ROUTING_MGR, FSM_DIRECTING_MGR
  - LOG_MESSAGE_EXCEPT from FSM_ROUTING_DIRECTING_MGR
  - MESSAGE_TO_OPERATOR from any routing and directing finite-state machines needing to report an exception on an exception
- Signals from distribution transport finite-state machines:
  - SEND_MU_EXCEPT from FSM_SEND_MGR
  - LOG_MESSAGE_EXCEPT from FSM_RECEIVE_MGR, FSM_SEND_MGR
  - MESSAGE_TO_OPERATOR from any distribution transport finite-state machines needing to report an exception on an exception
- Signals from other applications (UPM):
  - SEND_RELEASE_HOLD from UPM_MSG_FROM_OPERATOR
  - RCV_RELEASE_HOLD from UPM_MSG_FROM_OPERATOR
- Signals from lower-level operations finite-state machines:
  - RETRIABLE_EXCEPT from FSM_EXCEPT_TYPE
  - NONRETRIABLE_EXCEPT from FSM_EXCEPT_TYPE
  - QUEUE_CONTROL_COMPLETE from FSM_QUEUE_CONTROL
  - LOGGING_OK from FSM_LOG
  - MESSAGE_OK from FSM_MESSAGE
  - REPORT_COMPLETE from FSM_REPORT
- Signals from finite-state machines providing common services:
  - QUEUE_OK from QUEUE_MGR
  - QUEUE_NOT_OK from QUEUE_MGR
  - SCHED_FUNCTION_OK from FSM_SCHED_MGR
  - SCHED_FUNCTION_NOT_OK from FSM_SCHED_MGR
  - OBJECT_OK from SERVER_MGR
  - OBJECT NOT OK from SERVER_MGR

Inputs	States									
	RESET	QUEUE CTRL	EXPT TYPE	RELQ	DEQ	DEC LOCK	REPT	LOG	MSG	START TRAN
	01	02	03	04	05	06	07	08	09	10
SEND MU EXCEPTION	2b	/	/	/	/	/	/	/	/	/
LOG MESSAGE EXCEPTION	8c	/	/	/	/	/	/	/	/	/
ROUTING DIRECTING EXCEPTION	7e	/	/	/	/	/	/	/	/	/
MESSAGE TO OPERATOR	9d	/	/	/	/	/	/	/	/	/
SEND RELEASE HOLD	10h	/	/	/	/	/	/	/	/	/
RCV RELEASE HOLD	10i	/	/	/	/	/	/	/	/	/
RETRIABLE EXCEPTION	/	/	4k	/	/	/	/	/	/	/
NONRETRIABLE EXCEPTION	/	/	5f	/	/	/	/	/	/	/
REPORT COMPLETE	/	/	/	/	/	/	8c	/	/	/
QUEUE CONTROL COMPLETE	/	3a	/	/	/	/	/	/	/	/
LOGGING OK	/	/	/	/	/	/	/	9d	/	/
MESSAGE OK	/	/	/	/	/	/	/	/	1g	/
QUEUE OK	/	/	/	8c	6m	/	/	/	/	/
QUEUE NOT OK	/	/	/	8c	6m	/	/	/	/	/
SCHED FUNCTION OK	/	/	/	/	/	/	/	/	/	1g
SCHED FUNCTION NOT OK	/	/	/	/	/	/	/	/	/	1j
OBJECT OK	/	/	/	/	/	7e	/	/	/	/
OBJECT NOT OK	/	/	/	/	/	7e	/	/	/	/

Output Code	Function
a	Signal FSM_EXCEPT_TYPE with EXCEPTION_CODE to determine if the exception is retriable or nonretriable.
b	Signal FSM_QUEUE_CONTROL with SET_QUEUE_CONTROL to determine if the queues for the connection should be held, and, if so, then setting the hold.
c	Signal FSM_LOG with LOG_EXCEPT to log the appropriate operands for the exception.
d	Signal FSM_MESSAGE with DISPLAY_MESSAGE to display an exception message to the operator.
e	Signal FSM_REPORT with GENERATE_REPORT to generate any requested asynchronous reports.
f	Signal QUEUE_MGR with DEQ to remove the distribution from the next-DSU queue. The exception was determined to be nonretriable.
g	Signal calling machine with OPERATIONS_COMPLETE to indicate that the appropriate operations functions have been completed. These may not have been completed successfully.
h	Signal FSM_SCHED_MGR with OPR_START_SEND to indicate that an operator has requested that the send side be released and a DS_Send transaction program be scheduled.
i	Signal FSM_SCHED_MGR with OPR_START_RECEIVE to indicate that an operator has requested that DS_Receive be scheduled so that the flow of DMUs can be resumed from DS_Send.
j	Signal calling machine with OPERATIONS_NOT_OK to indicate that the appropriate operations functions have not been completed successfully.
k	Signal QUEUE_MGR with RELEASEQ to remove the in-use mark from the entry on the next-DSU queue. The exception was determined to be retriable and, following the RELEASEQ, the entry will be available to DS_Send for processing.
m	Signal SERVER_MGR with DECREMENT_OBJ_LOCK to decrement the DS lock count on the server object. If no server object exists, SERVER_MGR returns OBJECT_OK.

FSM_EXCEPT_TYPE

Determines if an exception is retriable or nonretriable. The request for this function is from FSM_OPERATIONS_MGR, and the exception information is passed for analysis.

FSM_EXCEPT_TYPE determines from the exception information if the DMU can possibly be transmitted at a later time, and, if so, signals FSM_OPERATIONS_MGR with RETRIABLE_EXCEPT. If the exception is such that something in the DMU is incorrect, and will never successfully be sent until the DMU changes (for example, the server name), FSM_EXCEPT_TYPE signals FSM_OPERATIONS_MGR with NONRETRIABLE_EXCEPT.

For retriable exceptions, FSM_EXCEPT_TYPE logs the DMU unique identifier in a retry log. On subsequent exceptions, FSM_EXCEPT_TYPE searches the retry log, and if the exception is tried and fails some specified number of times, FSM_EXCEPT_TYPE signals FSM_OPERATIONS_MGR with NONRETRIABLE_EXCEPT.

FSM_QUEUE_CONTROL

This machine determines if the next-DSU queues for the connection should be held based on the exception encountered, and, if so, sets the exception-hold indicator. The request for this function is from FSM_OPERATIONS_MGR and the exception information is passed.

FSM_QUEUE_CONTROL examines the exception, and determines if it is reasonable that other DMUs can successfully be transmitted on the connection specified. If so, the exception-hold is not set, and this machine signals FSM_OPERATIONS_MGR with QUEUE_CONTROL_COMPLETE. An example of this is an unknown server name that flows in the DMU. This exception does not necessarily affect the next DMU that may flow.

If FSM_QUEUE_CONTROL examines the exception and determines that it is likely that more DMUs will fail, FSM_QUEUE_CONTROL sets the exception-hold indicator on the queues for the specified connection in a table containing information about these queues. FSM_QUEUE_CONTROL signals FSM_OPERATIONS_MGR with QUEUE_CONTROL_COMPLETE. An example of an exception that implies that other DMUs will likely fail is an out-of-space condition on the receiver side.

FSM_MESSAGE

This machine sends a message to the operator concerning the current exception condition. The request for this function is from FSM_OPERATIONS_MGR or FSM_REPORT signalling DISPLAY_MESSAGE. The exception information needed to format the message is passed.

FSM_MESSAGE formats the message, then sends the message by signalling UPM_MSG_TO_OPERATOR. FSM_MESSAGE signals the calling machine with MESSAGE_OK.

FSM_REPORT

This machine generates a distribution report for exceptions encountered in DS_Router_Director, or exceptions in DS_Send that are nonretriable. The request for this function is from FSM_OPERATIONS_MGR signalling GENERATE_REPORT. The exception information plus the distribution on which the exception occurred is passed. This includes the list of exception destinations.

FSM_REPORT manages the following:

- Issuing WRITEQ to put the distribution on the router-director queue.
- Issuing the request to schedule the transaction DS_Router_Director.
- Issuing RELEASEQ to remove the in-use mark from the queue entry on the router-director queue to make it available for processing.

If exceptions occur in the queue or scheduling function, FSM_REPORT signals FSM_LOG to log the failure, and FSM_MESSAGE to notify the operator. In the case of a scheduling exception, FSM_REPORT also signals QUEUE_MGR with DEQ to remove the entry from the router-director queue.

In addition to the above, if FSM_OPERATIONS_MGR was signalled by FSM_DIRECTING_MGR and if the distribution could not be enqueued for delivery to any of the local destinations, then SERVER_MGR is signalled with BACKOUT, and any returned server report is queued for the destination agent.

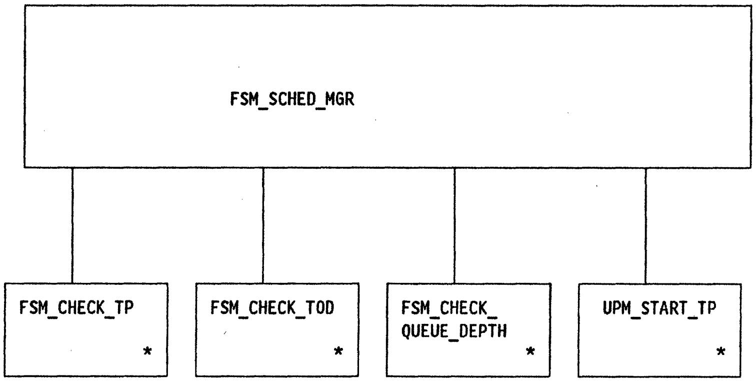
FSM_REPORT signals FSM_OPERATIONS_MGR with REPORT_COMPLETE in any case.

FSM_LOG

This machine logs the exception in the DMU exception log. The request for this function is from FSM_OPERATIONS_MGR or from FSM_REPORT signalling LOG_EXCEPT. The exception information is passed. The information logged for a given condition is listed in Appendix E.

FSM_LOG signals the calling machine with LOGGING_OK.

Scheduler



* indicates those finite-state machines not formally specified.

For more details regarding the finite-state machines, see:

- "FSM_SCHED_MGR" on page 351
- "FSM_CHECK_TP" on page 354
- "FSM_CHECK_TOD" on page 354
- "FSM_CHECK_QUEUE_DEPTH" on page 354
- "UPM_START_TP" on page 354

Figure 52. Scheduler Manager FSM Hierarchy

FSM_SCHED_MGR

The scheduler manager issues START_TRANSACTION to UPM_START_TP for any DS transaction program and for any destination agent program specified. The signalling machine passes the name of the transaction program to start and the queue identifier, if appropriate.

FSM_SCHED_MGR processes the following signals:

- OPR_START_RECEIVE to start DS_Receive.
- OPR_START_SEND to start DS_Send.
- START_REQUEST to start:
  - DS_Router_Director.
  - An agent.
  - DS_Send.

    For DS_Send, one or both of the following occur:
    - FSM_SCHED_MGR signals FSM_CHECK_TOD to determine if there is any scheduling by time of day for the next-DSU queue. If there is, the time of day is specified on the START_TRANSACTION and the system resource manager starts the transaction program at the specified time.
    - If no time of day is specified, FSM_SCHED_MGR signals FSM_CHECK_QUEUE_DEPTH to determine if there is a queue depth trigger for starting DS_Send for the specified next-DSU queue. If the queue depth has been reached or no queue depth is set, FSM_SCHED_MGR issues START_TRANSACTION for DS_Send. If the queue depth has not been reached, this machine returns without issuing a START_TRANSACTION.

If START_TRANSACTION is not successful, FSM_SCHED_MGR signals the calling machine SCHED_FUNCTION_NOT_OK and passes the available exception information.

Function: This finite-state machine describes the functional processing for scheduling DS_Router_Director service transaction program, DS_Send service transaction program, DS_Receive service transaction program, or the destination agent. For a further description, see "FSM_SCHED_MGR" on page 351.

This FSM gets control from one of the following:

- Signals from presentation services, DS_Router_Director, DS_Send, DS_Receive:
  - START_REQUEST from FSM_LOCAL_SCHED, FSM_REMOTE_SCHED, FSM_RCV_ENQ_SCHED, DS_RCV_ENQ_SCHED
- Signals from finite-state machines providing common services:
  - OPR_START_SEND from FSM_OPERATIONS_MGR
  - OPR_START_RECEIVE from FSM_OPERATIONS_MGR
- Signals from lower-level finite-state machines:
  - DS_ROUTER_DIRECTOR from FSM_CHECK_TP
  - DS_SEND from FSM_CHECK_TP
  - AGENT from FSM_CHECK_TP
  - TOD_SET from FSM_CHECK_TOD
  - NO_TOD_SET from FSM_CHECK_TOD
  - QUEUE_DEPTH_REACHED from FSM_CHECK_QUEUE_DEPTH
  - NO_QUEUE_DEPTH_SET from FSM_CHECK_QUEUE_DEPTH
  - QUEUE_DEPTH_NOT_REACHED from FSM_CHECK_QUEUE_DEPTH
  - START_TRANSACTION_OK from UPM_START_TP
  - START_TRANSACTION_NOT_OK from UPM_START_TP

Inputs	**States**				
	RESET	**TP CHECK**	**TOD CHECK**	**QUEUE DEPTH**	**START PEND**
	01	**02**	**03**	**04**	**05**
START REQUEST	2a	/	/	/	/
OPR START SEND	5b	/	/	/	/
OPR START RECEIVE	5c	/	/	/	/
DS ROUTER DIRECTOR	/	5d	/	/	/
DS SEND	/	3e	/	/	/
AGENT	/	5f	/	/	/
TOD SET	/	/	5h	/	/
NO TOD SET	/	/	4i	/	/
QUEUE DEPTH REACHED	/	/	/	5b	/
QUEUE DEPTH NOT REACHED	/	/	/	1g	/
NO QUEUE DEPTH SET	/	/	/	5b	/
START TRANSACTION OK	/	/	/	/	1g
START TRANSACTION NOT OK	/	/	/	/	1j

Output Code	Function
a	Signal FSM_CHECK_TP with CHECK_TP to determine what transaction program is being requested.
b	Signal UPM_START_TP with START_TRANSACTION passing DS_SEND as the *target_tpn* and WHEN_SESSION_PREALLOCATED as the indicator of when the TP should be started.
c	Signal UPM_START_TP with START_TRANSACTION passing DS_RECEIVE as the *target_tpn* and WHEN_SESSION_PREALLOCATED as the indicator of when the TP should be started.
d	Signal UPM_START_TP with START_TRANSACTION passing DS_ROUTER_DIRECTOR as the *target_tpn* and NO_PREALLOCATION_REQUIRED as the indicator of when the TP should be started.
e	Signal FSM_CHECK_TOD with CHECK_TOD to determine if time of day scheduling has been defined for this next-DSU queue.
f	Signal UPM_START_TP with START_TRANSACTION passing the agent as the *target_tpn* and NO_PREALLOCATION_REQUIRED as the indicator of when the TP should be started.
g	Signal calling machine with SCHED_FUNCTION_OK to indicate that the scheduling request has been made if a transaction program is to be started, or that currently no TP is to be started.
h	Signal UPM_START_TP with START_TRANSACTION passing DS_SEND as the *target_tpn* and WHEN_SESSION_PREALLOCATED and the time of day as indicators of when the TP should be started.
i	Signal FSM_CHECK_QUEUE_DEPTH with CHECK_QUEUE_DEPTH to determine if the scheduled queue depth has been reached to indicate that DS_Send should be scheduled.
j	Signal calling machine with SCHED_FUNCTION_NOT_OK to indicate that the scheduling request did not receive an OK return code.

FSM_CHECK_TP

Determines the transaction program name to be started. The transaction program name passed to FSM_SCHED_MGR is passed to this machine with the signal CHECK_TP.

The output of this machine is one of the following signals to FSM_SCHED_MGR:

- DS_ROUTER_DIRECTOR
- DS_SEND
- AGENT

  For the signal AGENT, the transaction program name is passed.

FSM_CHECK_TOD

Determines if any time of day scheduling is defined for the specified LU name, and mode name, and, if so, passes the time of day to FSM_SCHED_MGR. The request for this function is from FSM_SCHED_MGR.

This machine examines a table that contains attributes for the LU name, mode name passed. If a time of day list is defined, the next time of day following the current time is chosen, FSM_CHECK_TOD signals FSM_SCHED_MGR with TOD_SET and passes the time of day.

If no time of day is defined, FSM_CHECK_TOD signals FSM_SCHED_MGR with NO_TOD_SET.

FSM_CHECK_QUEUE_DEPTH

Determines if queue-depth scheduling is defined for the specified LU name, and mode name, and if so, checks whether the queue depth has been reached. The request for this function is from FSM_SCHED_MGR.

This machine examines a table that contains attributes for the *LU name*, *mode name* passed to this machine. If a schedule queue depth is defined, it is compared to the current queue depth. If the current queue depth is equal to or greater than the schedule queue depth, FSM_CHECK_QUEUE_DEPTH signals FSM_SCHED_MGR with QUEUE_DEPTH_REACHED. If not, this machine signals QUEUE_DEPTH_NOT_REACHED.

If a schedule queue depth is not defined, this machine signals NO_QUEUE_DEPTH_SET.

UPM_START_TP

This is an undefined protocol machine. It starts a transaction program. This function is embedded in the local execution environment of DS.

SERVER_MGR

The server manager controls access to the server object. All the machines needing access to the server object send signals representing server verbs to the server manager, and receive back an indication of the success or failure of the requested operation. The signals back may or may not have data associated with them. The signals to the server manager have the server object information from the distribution. The name of the server is assumed to be passed to the server manager along with the other associated parameters. The input signals are handled as follows:

- INCREMENT_OBJ_LOCK

  The server manager checks to determine whether a server object exists. If not, the server manager signals back OK. If a server object does exist, the server manager inspects the DS or agent (as appropriate) lock count, and if it is 0, issues an Assign_Read_Access server verb and sets the appropriate lock count to 1. Otherwise, it simply increments the DS or agent (as requested) lock count. The server manager then reports the results of the operations and returns parameters to the signalling machine. The state of the server object after the successful completion of the verb is read-locked.

- READ

  The server manager issues the Read verb or an Initiate_Read/Read sequence of verbs (as appropriate) with the supplied parameters to the appropriate server. The server manager reports the results and returns parameters to the calling machine. The possible return codes are OBJECT_OK (indicating the successful completion of the verb and return of server object data), NO_OBJECT_EXISTS (indicating that the distribution has no server object), OBJECT_EOD (indicating that no more server object data remains) and OBJECT_NOT_OK.

  The server return code of SPECIFIC_SERVER_EXCEPTION is mapped to OBJECT_NOT_OK. In this case, distribution reports are suppressed, and any specific server reports supplied by the server are delivered to the origin agent.

- WRITE

  The server manager issues the Write verb or an Initiate_Write/Write sequence of verbs with the supplied parameters to the appropriate server. The server manager reports the results and returns parameters to the signalling machine. The possible return coaes are OBJECT_OK, OBJECT_NOT_OK and SPECIFIC_SERVER_EXCEPTION.

- TERMINATE_READ

  The server manager issues the Terminate_Read verb (if appropriate) with the parameters supplied with the signal into the server manager machine. The server manager reports the results and returns parameters to the signalling machine. The possible return codes are OBJECT_OK and OBJECT_NOT_OK. If no server object exists, or if no Initiate_Read verb was issued for the server object previously, the Terminate_Read verb is suppressed and the SERVER_MGR simply returns OBJECT_OK.

- TERMINATE_RESTARTABILITY

  The server manager issues the Terminate_Restartability verb (if appropriate) with the parameters supplied with the signal into the server manager machine. The server manager reports the results and returns parameters to the signalling machine. The possible return codes are OBJECT_OK and OBJECT_NOT_OK. If no server object exists, or if restart capability was not specified in the Initiate_Read or Initiate_Write verb, the Terminate_Restartability verb is suppressed and the SERVER_MGR simply returns OBJECT_OK.

- TERMINATE_WRITE

  The server manager issues the Terminate_Write verb (if appropriate) with the parameters supplied with the signal into the server manager machine. The server will assign read access to DS upon completion of the Terminate_Write, and server manager sets the DS lock count to 1. The server manager reports the results and returns parameters to the signalling machine. The possible return codes are OBJECT_OK and OBJECT_NOT_OK. If no server object exists, or if no Initiate_Write verb was issued for the server object previously, the Terminate_Write verb is suppressed and the SERVER_MGR simply returns OBJECT_OK.

- DECREMENT_OBJ_LOCK

  The server manager checks the server object information in the distribution to determine if a server object exists. If no server object exists, the server manager signals back OK.

  If a server object exists, the server manager decrements the DS or agent (as appropriate) lock count. If the agent lock count is decremented from 1 to 0, the server manager issues a Release_Read_Access verb for the agent. If, after the decrementing operation, both the DS and agent lock counts are 0, the server manager issues a Release_Read_Access verb for DS. The server manager then reports the results, and returns parameters to the signalling machine.

- BACKOUT

  This signal causes SERVER_MGR to issue a Backout_Server_Object server verb. Any returned server report is passed back to the caller. OBJECT_OK is returned to the caller in any event.

- QUERY_LAST_BYTE_RCVD

  This signal causes SERVER_MGR to issue a Query_Last_Byte_Received server verb. The byte count returned by the server is passed back to the caller to be included in a CRMU(SUSPENDED). OBJECT_OK or OBJECT_NOT_OK is returned, as appropriate.

Output signals

- OBJECT_OK

  The requested operation was performed successfully, or was not necessary.

- NO_OBJECT_EXISTS

  This return code indicates that a READ operation has been issued for a distribution which contains no server object. This condition will be detected on the initial READ for the distribution.

- OBJECT_EOD

  This return code indicates that a Read verb has encountered the "end-of-data" condition for the server object. Previous Read verbs have already exhausted the server object data, and no data remains to be returned to the caller when this condition is detected.

- OBJECT_NOT_OK

  The requested operation failed. On READ operations, SPECIFIC_SERVER_EXCEPTION is mapped to this return code. In this case, distribution reports are suppressed, and any specific server reports supplied by the server are delivered to the origin agent.

- SPECIFIC_SERVER_EXCEPTION

  This is returned only on WRITE operations. The specific server accepted the server object before receiving the complete object.

QUEUE_MGR

The QUEUE_MGR provides a common service for all the FSMs in the DSU for controlling the router-director queue, the local delivery queues, the next-DSU queues, the control MU queue, and the mid-MU restart queue. It also increments the queue depth (a scheduling mechanism) for each of the next-DSU queues when an element is enqueued, and resets the queue depth when the queue is empty or held. It takes as input queue-operation signals and associated supplied parameters, and returns the success or failure of the operation as well as the associated output parameters. The input signals are:

- HOLD

  The queue manager holds the designated queues, so that subsequent READQ signals return QUEUE_EMPTY. The *type* of hold to be set must be supplied:

  - *Exception*-holds are set automatically when a retriable exception is encountered, and cleared automatically by the first READQ of a new instance of DS_Send.
  - *Operator*-holds are set and cleared only by explicit operator action.

  If the HOLD is successful, QUEUE_OK is returned; if the HOLD fails, QUEUE_NOT_OK is returned.

- DEQ

  The queue manager removes the distribution from the queue and returns it to the machine issuing the signal. The data accompanying this signal includes the name of the queue, and the identifier of the distribution to be dequeued. If the distribution is dequeued successfully, QUEUE_OK is returned; if the dequeueing operation fails, QUEUE_NOT_OK is returned.

- WRITEQ

  The queue manager places the distribution at the end of the queue. The data accompanying this signal includes the name of the queue and the entry to be enqueued. This function marks the queue entry (the distribution) as being *in-use*. The distribution cannot be read when it is marked in-use. If the distribution is successfully written to the queue, QUEUE_OK is returned; if the write operation fails, QUEUE_NOT_OK is returned.

- READQ

  The queue manager reads the distribution on the queue, but does not remove it from the queue. The data accompanying this signal include the name of the queue and the identifier of the distribution, or an indication that the "next available" distribution is to be read. The first READQ for the next-DSU queues issued by DS_Send will clear an exception-hold indicator, if it is set on. On subsequent READQ signals, QUEUE_EMPTY is returned if either the exception- or operator-hold indicator is *on*.

  If the "next available" distribution is to be read, the distributions available for transmission to the partner are scanned, and the highest priority distribution is chosen. If no available distributions are on the queue (the queue may be empty, all distributions may be in-use, or the queue may be held), READQ returns QUEUE_EMPTY.

  If the request specifies a particular distribution, but the distribution's in-use mark is set, QUEUE_ENTRY_IN_USE is returned unless *suspend* is specified. A caller specifying a specific distribution with *suspend* is suspended for a period of time if the distribution is in-use. If the distribution becomes available while the caller is suspended, the caller's READQ completes successfully; if the distribution does not become available, QUEUE_ENTRY_IN_USE is returned.

  Otherwise, the output of this function is the information from the distribution and a return code of QUEUE_OK. This function marks the queue entry (the distribution) as being in-use. If the operation fails, QUEUE_NOT_OK is returned.

- RELEASEQ

  The queue manager removes the in-use mark from the distribution. The data accompanying this signal includes the queue name and the identifier of the distribution. If the in-use mark is successfully removed, QUEUE_OK is returned; if not, QUEUE_NOT_OK is returned.

Output signals

- QUEUE_OK

  The requested operation was performed successfully.

- QUEUE_NOT_OK

  This return code indicates that the requested operation failed because of an exception condition in the queue service.

- QUEUE_ENTRY_IN_USE

  This return code indicates that a particular distribution could not be read because its in-use mark is set.

- QUEUE_EMPTY

  This return code indicates that no queue entry satisfying the READQ's search criteria can be found.

BUILDER

The transport services MU builder encodes FS1 and FS2 DMUs, REMUs, SEMUs, CRMUs, CQMUs and PRMUs. The builder is signalled by:

- DS_SEND_BUILD_SEND_DMU and DS_SEND_SEND_DMU_NO_MU_ID with BUILD and END_OBJECT to build FS2 DMUs.
- DS_SEND_CLEANUP_EXCEPT and DS_SEND_ISSUE_SEMU_ON_CRMU to build FS2 SEMUs.
- DS_RCV_SEND_ERR_REMU, DS_RCV_SEND_ERR_SUSP_TERM_REMU and DS_RCV_REMU_SUSP_TERM to build FS2 REMUs.
- DS_RCV_SEND_ERR_CRMU, DS_RCV_SEMU_HANDLER, DS_RCV_ENQ_SCHED and DS_RCV_CQMU_HANDLER to build FS2 CRMUs.
- DS_SEND_RELEASE_ON_CRMU to build FS2 CQMUs.
- DS_SEND_RETRY_ON_REMU, DS_SEND_TERMINATE_DIST, DS_SEND_PURGE_ON_CRMU and DS_SEND_RETRY_ON_CRMU to build FS2 PRMUs.
- FSM_DIST_ENCODE_CONTROL with BUILD and END_OBJECT to build FS1 DMUs.
- FSM_SEMU_ENCODE, with BUILD_SEMU to build FS1 SEMUs.
- FSM_REMU_ENCODE with BUILD_REMU to build FS1 REMUs.

The builder encodes the MUs as defined in Appendix G. To indicate that the builder has successfully encoded a portion of the MU, this machine signals BUILD_OK. To indicate that the next portion of the MU to be sent on the conversation contains part of the data server object, this machine signals BUILD_OK_GET_OBJECT. When the builder has completed the encoding of an MU, this machine signals BUILD_COMPLETE.

The information needed to encode the MUs (for example, buffer of data from reading the server object, exception information for either the SEMU or the REMU) is passed to the builder.

Exceptions from the builder cause the signal BUILD_NOT_OK and information about the exception to be returned to the caller.

PARSER

The transport services MU parser decodes FS1 and FS2 DMUs, REMUs, CRMUs, CQMUs and PRMUs. The parser is signalled by:

- DS_RCV_RECEIVE_DMU and DS_RCV_RECEIVE_DMU_NO_MU_ID with PARSE to decode FS2 DMUs.
- DS_SEND_RECEIVING with PARSE to decode FS2 CRMUs and REMUs.
- PREPARSER to decode FS2 CQMUs, PRMUs and SEMUs.
- FSM_DIST_DECODE_CONTROL with DECODE to decode FS1 DMUs.
- FSM_SEMU_DECODE with PARSE_SEMU to decode FS1 SEMUs.
- FSM_REMU_DECODE with DECODE_REMU to decode FS1 REMUs.

The parser decodes the MUs as defined in Appendix G and builds a control block containing the MU's information. The parser may return the following return codes:

- PARSE_OK

  The MU passed to the parser is a DMU, and the parser successfully decoded the portion of the DMU passed to it.

- PARSE_OK_OBJECT

  The portion of the DMU passed to the parser contains a portion of the server object, which should be passed to the server.

- PARSE_COMPLETE

  The DMU passed to the parser was successfully and completely decoded.

- PARSE_NOT_OK

  An exception was detected while decoding the MU.

- CRMU

  The MU passed to the parser is a CRMU, which was successfully parsed.

- REMU

  The MU passed to the parser is a REMU, which was successfully parsed.

Appendix A. Acronyms and Abbreviations

API	Application Program Interface
APPC	Advanced Program-to-Program Communication
BCPB	Basic Conversation Protocol Boundary
CGCSGID	Coded Graphic Character Set Global Identifier
CMU	Control Message Unit
CQMU	Completion Query Message Unit
CRMU	Completion Report Message Unit
DCMU	Distribution Continuation Message Unit
DEN	Distribution Element Name
DGN	Distribution Group Name
DIA	Document Interchange Architecture
DMU	Distribution Message Unit
DRMU	Distribution Report Message Unit
DS	Distribution Services
DSU	Distribution Service Unit
DTM	Date/Time
DTMU	Distribution Transport Message Unit
ECS	Enhanced Character String
FIFO	First-In, First-Out
FSM	Finite-State Machine
FS1	Format Set 1
FS2	Format Set 2
GEN	General Server
GMT	Greenwich Mean Time
IRN	(Path Control) Intermediate Routing Node
LU	Logical Unit
MB	Megabytes
MU	Message Unit
NAU	Network Addressable Unit
PB	Protocol Boundary
PC	Path Control
PCIRN	Path Control Intermediate Routing Node

PRMU	Purge Report Message Unit
PS	Presentation Services
RBP	Restart Byte Position
RAMU	Reset Accepted Message Unit
REMU	Receiver Exception Message Unit
REN	Routing Element Name
RGN	Routing Group Name
RRMU	Reset Request Message Unit
SEMU	Sender Exception Message Unit
SNA	Systems Network Architecture
SNACR	SNA Condition Report
SNA/DS	SNA/Distribution Services
TP	Transaction Program
TPN	Transaction Program Name
TPF	Transmission Priority Field
UPM	Undefined Protocol Machine

Appendix B. Introduction to Finite-State Machines

Introduction to FSMs

A finite-state machine (FSM) is represented as a *state-transition matrix* consisting of a set of *states*, a set of *input signals*, a set of *output codes*, a *next state function*, and an *output function*. The states are a small number of named values (the *state names*). An FSM's execution depends on a combination of processing and memory, where the memory consists of one of the states of the FSM. The processing is defined by the state-transition matrix, each *cell* (row-column intersection) of which shows the new state of the FSM, and the output code to be executed. Within this matrix, each state is given a number, as well as a name, for notational convenience.

The syntax of the state-transition matrix is shown in Figure 53 on page 364. The column headings give the FSM state names, while the row headings give the input signals. The matrix cells define the state transitions and output codes when the FSM is in the given states and receives the given input signals.

If the next-state indicator is a number *n*, the FSM enters state *n*. If the next-state indicator is a cannot-occur indicator (/), this is an execution-time error: calls of the FSM cannot encounter this indicator because previous logic has filtered out the input for that state of the FSM. If the next-state indicator is a no-state-change indicator (-), the FSM remains in its same state.

In the state-transition matrix, double horizontal lines are used to group input lines together to improve readability. Their location has no bearing on the FSM function. For compactness, mnemonic abbreviations are used in the matrices.

An FSM comes into existence initialized to state 1. If another state is to be the initial state, the FSM is initialized explicitly by *signalling* the FSM with an appropriate input signal.

An FSM always has at least one state and one input signal. Consequently, the state-transition matrix always has at least one row and one column. Output codes may be absent entirely.

fname

	States	
	sna	sna
Inputs	snu	snu
ic	ac	ac
ic	ac	ac
ic	ac	ac

Output Code	Function
oc	Output logic statements
oc	Output logic statements

Legend:

fname = FSM name
sna = state name
snu = state number
ic = input signal name
ac = action code
oc = output code

An action code (ac) has the following possible syntax:

n normal state transition to state *n* (corresponding to some snu)

*n*oc normal state transition to state *n* (corresponding to some snu)
and execution of the function identified by oc

- same-state transition (remain in the same state)

-oc same-state transition (remain in the same state)
and execution of the function identified by oc

/ "cannot occur" condition, no state change

Figure 53. Syntax of an FSM State-Transition Matrix and Output Codes

Appendix C. Implementation Alternatives

Because an architecture can be implemented in equipment of different sizes to serve various application purposes, it is not unusual for a given implementation to choose to implement less than the complete architecture. Certain functions can be omitted without harming the overall network; others cannot. Therefore, the architecture defines the rules under which implementations may choose to omit functions.

The effect of these choices upon the total size of the implementation can be very significant. The smallest compliant implementation might be about 1/50 the size of the largest.

References to these choices will be found in other parts of this document. Anyone interested only in a general understanding of the architecture can safely ignore such references and need not read this appendix. Anyone interested in designing an implementation of the architecture should read and thoroughly understand this appendix.

Categories of Choices

There are six categories of choices. In decreasing order of importance, they are:

1. Protocol boundary exposure
2. Role
3. Base and option sets
4. Electives
5. Specializations
6. Optimizations (no observable function omitted)

Every implementation makes choices in each of the categories. A choice made in one category, or a combination of choices in several categories, may often restrict the possible choices in other categories.

Implementations may have an internal structure that differs from the architecture model. Provided this choice of internal structure has no externally perceived effects, it is not subject to architectural compliance rules and is considered an optimization. Some optimizations that are especially appropriate in highly specialized implementations are mentioned in this document. However, they are intended only as helpful suggestions.

Protocol Boundary Exposure

The Choice of Open or Closed

Implementations have either open or closed protocol boundaries (PBs). Typically, an open PB is implemented as an application program interface (API). Alternatively, some PB functions can be implemented as an end-user interface--in other words, in the form of screens and keyboard inputs. The latter form of PB is suitable for operations verbs but is not suitable for Send_Distribution or Receive_Distribution: these verbs require sequence numbers and may contain encoded agent objects.

Open agent and server protocol boundaries allow any agents and servers to interact with DS. Assuming that the user, agent, and server names are known by the DS network in one or more DSUs, any distribution request, within the base set of functions described below, will be successfully serviced.

Implementations not offering an open protocol boundary may, depending on other choices, find their implementation responsibilities significantly reduced. Such implementations have a *closed* protocol boundary, since they provide only a restricted set of services to restricted sets of users, agents, and servers.

The operations protocol boundary cannot be closed. Whatever operations functions are supported by an implementation are available, by one means or another, to the operations and maintenance staff serving the network as a whole. For example, a central operator at one DSU might obtain operations information from another DSU by telephoning the operator at that DSU who obtains the desired information by means of an interactive end-user interface. Unattended DSUs may have remote interactive end-user interfaces, or if they are expected to require limited operational interaction, they may use simpler remote techniques such as downloading their software or microcode and retrieving their logs and dumps. In other words, an open operations PB means that an operator, not necessarily at the DSU, must be able to control the DSU's operation by means that are appropriate for the frequency of required interventions.

The agent protocol boundary and the server protocol boundary are either both open or both closed.

Rules for Closed Protocol Boundaries

Implementations with closed protocol boundaries:

- May limit their originating and delivering functions.
  - May restrict their set of users to 0.
  - May restrict their set of agents to 1.
  - May restrict the services provided to only those needed by the restricted set of users or agents.
  - May restrict their support to only one object, agent, or server.
  - When supporting server objects may restrict their set of server names to 1, which could be the default (general server).

- Never generate any protocol on behalf of their privately integrated application that an equivalent application using an open protocol boundary cannot generate using the defined PB verbs.
- Are able to exchange traffic with equivalent applications using open protocol boundaries, regardless of whether the communication is direct or via intermediate DSUs. Closed PB implementations need not implement function equivalent to any PB parameters other than whatever is necessary to comply with this rule.
- Are able to exchange traffic with copies of themselves, regardless of whether the communication is direct or via intermediate DSUs.
- Are able to interconnect with any other DS implementation. (Interconnection to older FS1-only implementations requires support of the FS1 option subset.)
  - Are able to send MUs requiring base services through any intermediate FS2 DSUs.
  - Are able to send MUs requiring optional services through any intermediate FS2 DSUs that provide the required services.

Rules for Open Protocol Boundaries

Implementations with open protocol boundaries:

- Support the base semantics of both agent and server PBs.
- Support, for each optional subset they choose to implement, whatever additional PB semantics the subset requires.
- Do not restrict their originating and delivering function.
  - Support as many users as their largest equipment configuration can reasonably support.
  - Support a variety of architecturally-defined and installation-defined agents.
  - Support a variety of architecturally-defined and installation-defined servers.
  - Contain installation-defined tables or lists against which the destination agent and destination server names in the incoming MU can be compared.
  - Originate or accept for delivery any distribution requiring services defined in the base or any chosen option sets that specifies user, agent, and server names defined for the DSU.
- Are able to interconnect with any other DS implementation. (Interconnection to older FS1-only implementations requires support of the FS1 option subset.)

Role

Implementations may limit their DSUs to certain roles only. The choices are:

1. Origin-only: closed PB only
2. Destination-only: closed PB only
3. Origin and Destination (End-only): open or closed PB
4. Intermediate-only: closed PB (no application program interface at all)
5. All roles: open or closed PB

Combinations such as origin and intermediate are possible but too unlikely to justify discussion. The end-only and all-roles combinations are the most usual. The rules for the elementary roles are given below. Rules for combinations are the union of the rules for the appropriate elementary roles.

Rules for Origin Role

- Origin role implementations with a closed PB and appropriately specialized applications or devices, such as a card reader, may restrict themselves to only originating distributions.
- Implementations that provide an open protocol boundary are able to process distribution-originating verbs passed across the agent PB. Open PB implementations also support the receiving verbs and, therefore, cannot be origin-only.
- Implementations with closed protocol boundaries may limit the types of traffic they originate, but they never generate MUs that other DSUs cannot process normally.
- Implementations are able to generate distribution reports, unless they have a closed PB and are specialized in traffic that never requests reporting.

Rules for Destination Role

- Destination role implementations with a closed PB and appropriately specialized applications or devices, such as a line printer, may restrict themselves to serving only as destinations.
- Implementations that provide an open protocol boundary are able to process distribution-delivery verbs passed across the agent PB. Open PB implementations also support the sending verbs and therefore cannot be destination-only.
- An implementation in the destination role accepts any MU requiring base or selected option set function that specifies a known destination agent and server, stores the object using the specified server, and delivers the distribution to the specified agent by making an entry in the appropriate queue. Then, if possible, it initiates the specified destination agent.
- Implementations in the destination role reject any MUs containing unrecognized structures within parents that preclude them but accept and ignore unrecognized structures within parents that allow them.

- Implementations in the destination role reject MUs containing any flag bits they don′t understand in the first two bytes of the distribution flags.
- Implementations in the destination role reject MUs that do not meet the minimum parsing checks.
- Implementations in the destination role may reject MUs that do not meet optional parsing checks.
- Implementations in the destination role reject MUs containing any service parameters they don′t understand.
- Implementations in the destination role process the distribution in a manner consistent with the specified service parameters.
- Implementations are able to generate distribution reports, unless they have a closed PB and are specialized in traffic that never requests reporting.

Rules for Intermediate-only Role

- Implementations in the intermediate role completely receive an MU, and depending on the integrity and protection specified for the distribution, store it and any objects it may contain before forwarding it.
- When an intermediate node receives a multiple-destination distribution whose destinations are reached over different outgoing connections, implementations generate multiple outgoing MUs, each containing byte-perfect copies of the objects and a portion of the received destination list.
- Implementations in the intermediate role reject MUs that do not meet the minimum parsing checks.
- Implementations in the intermediate role may reject MUs that do not meet optional parsing checks.
- Implementations in the intermediate role reject MUs specifying service parameters that they cannot provide.
- Implementations in the intermediate role process the distribution in a manner consistent with the specified service parameters.
- Implementations in the intermediate role pass through, unchanged, any unrecognized structures occurring within parent structures where unrecognized children are allowed.
- Implementations in the intermediate role reject MUs containing non-understood flag bits in the first byte of the distribution flags.
- Implementations are able to generate distribution reports, unless they have a closed PB and are specialized in traffic that never requests reporting.

Base and Option Sets of Functions

In DS, as in other architectures, the notion exists of a mandatory or base set of the functions that all implementations support except insofar as their choices of PB exposure, role, and specialization may have made the functions inapplicable. Individual functions not included in the base set are options that implementations may select, but not on an individual function-by-function basis. The number of optional functions is so large that if implementations were free to select them individually, it is unlikely that any two implementations would select exactly the same set. As a result, connectivity between them would be impaired or impossible. To avoid this, the optional functions are grouped into named sets.

Base and Option Set Diagram

A convenient way to represent the option sets is with a diagram (see Figure 54). The complete set of functions described by the architecture is the union of all the rectangles in the diagram. The option sets that an implementation may omit are represented by smaller rectangles inside the full diagram. An option set that is located underneath another set is a prerequisite for the upper set and cannot be omitted unless all the sets above it are also omitted.

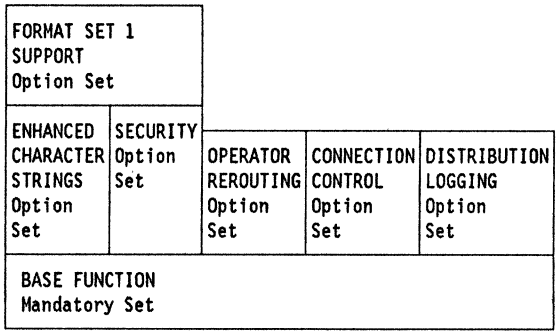

FORMAT SET 1 SUPPORT Option Set
ENHANCED CHARACTER STRINGS Option Set
SECURITY Option Set
OPERATOR REROUTING Option Set
CONNECTION CONTROL Option Set
DISTRIBUTION LOGGING Option Set
BASE FUNCTION Mandatory Set

Figure 54. Base and Option Set Diagram

General Rules for Base and Option Sets

1. DS implementations include all the functions defined in the base that apply to their choices of role, PB exposure, and specialization.
2. Each option set must be implemented completely.
3. Implementations never provide a function defined in a set in some way different from that defined by DS. Implementations always provide that function and all other functions in the set only as specified by the architecture.

Summaries of the major features of each set follow.

Rules for the Base Set

Base PB Verbs

- Implementations with open PBs accept both high- and basic-integrity verb sequences for sending and receiving distributions using the following verbs:
  - Send_Distribution
  - Query_Distribution_Sending
  - Sending_Sequence_Completed
  - Receive_Distribution
  - Receiving_Sequence_Completed
- Implementations with open PBs receive reports from either the distribution service or from the server using the following verbs:
  - Receive_Distribution_Report
  - Obtain_Local_Server_Report
- Implementations with open PBs interact with the server using the following verbs:
  - Assign_Read_Access
  - Release_Read_Access
  - Initiate_Read
  - Read
  - Terminate_Read
  - Initiate_Write
  - Write
  - Terminate_Write
  - Backout_Server_Object
- Implementations with open PBs perform the following checks on each verb at the protocol boundary when originating distributions:
  - All structures in the verb are checked for consistency with the length, contents, and conditions of presence as defined in Appendix F.
  - If an origin user is specified, the DSU checks that that user is local.
  - The destination list is examined and duplicate destinations are eliminated.
  - If reporting is requested, the DSU determines whether or not reports will be returned to it, and if so, checks that the report-to user and agent are local. If not, the DSU confirms that its local directory and routing table contain entries that allow it to honor its reporting responsibilities.
  - If a specific server is specified, the DSU checks that the server exists locally.

Base Service Parameters

Implementations are able to receive MUs and accept PB requests (for open PB implementations) with, and provide the required services for, the following service parameter specifications:

1. *priority*: REQUIRE_LEVEL_GE for any of 18 levels: FAST, CONTROL, DATA_16, ..., DATA_1. When sending on a given connection, implementations send all dis-

tributions of priority n before sending any of priority n - 1 (unless they have elected to fold the data priorities as described in "Electives" on page 377).

2. *protection*:

   a. REQUIRE_LEVEL_GE LEVEL2. Implementations store both the DS control information and the objects on non-volatile storage.
   b. REQUIRE_LEVEL_GE LEVEL1. Implementations may provide either LEVEL1 or LEVEL2 protection.

3. *capacity*:

   a. REQUIRE_LEVEL_GE ZERO. Implementations are able to selectively route based on this *capacity* service level. Network administrators may define DSUs that will not accept distributions with server objects provided that the routing tables of the surrounding DSUs are appropriately defined so that only zero-capacity distributions are routed to them.
   b. REQUIRE_LEVEL_GE 1MB, 4MB, 16MB. Base implementations in the intermediate role can, in normal operation, store and forward server objects of up to 16MB in size. Base implementations can selectively route on any of the three nonzero capacity service levels.

      Network administrators may define DSUs that will not accept distributions with server objects larger than either of the lower levels, 1MB or 4MB. If so, they define the routing tables of the surrounding DSUs so that distributions are routed appropriately.

      Implementations may optimize their storage management on the assumption that the server object will not exceed the specified size. At the PB the server object size is validated against the capacity requested only if *server_object_byte_count* is supplied. When receiving an MU, if the receiving process determines that the server object size exceeds the capacity requested, the DSU may either abort the transfer process at that point or attempt to receive it despite its size. If the distribution is successfully received, the capacity service parameter is set to the appropriate higher level before the distribution is forwarded.

4. *security*: REQUIRE_LEVEL_GE LEVEL1. LEVEL1 security requires no special action.

Base Reporting

Implementations are able to honor the reporting requirements for exception reporting.

Base Encoding Support

Implementations are able to send and receive the following types of FS2 message units:

- DTMU
- DCMU, except:
  - When sending, the restart position needs to be only the beginning of the LLID following the last LLID received (sender restarting within an object is an elective--Sender Byte-Count Restart elective).
  - When receiving, DCMUs that restart in mid-object may be rejected (receiver restarting within an object is an elective--Receiver Byte-Count Restart elective).
- DRMU

- CRMU, except that byte-position information need not be generated and may be ignored on receipt.
- CQMU
- PRMU
- SEMU
- REMU
- RRMU
- RAMU

Base Scheduling

- Implementations are able, subject to possible scheduling constraints, to initiate the SNA service transaction programs that perform the sending whenever distributions are ready to be sent.
- Implementations allow other DSUs to initiate either their receiving or sending SNA service transaction programs (DS_Receive or DS_Send) by means of an LU 6.2 Attach FM header.

Base Protocol

- Before sending any DMUs on a newly defined or discovered connection, the partner DSUs initialize MU-ID registries for the connection using an RRMU-RAMU exchange. On existing connections, RRMU-RAMU exchanges should be performed at least monthly.
- Each implementation supports the base FS2 protocol and gracefully tolerates any elective choices that differ from its own.
- Successful FS2 high-integrity transfers are completed by a CRMU-PRMU exchange.
- Unsuccessful transfers detected by the sender are indicated by the sender with a Send_Error followed by a SEMU.
- Unsuccessful transfers detected by the receiver are indicated by the receiver with a Send_Error followed by a REMU.
- Implementations support Mid-MU Restart at any of the highest-level LLID boundaries after the Dest_List, including LLID structures they do not recognize.

Base Routing and Directing

- Implementations maintain queues of traffic and requests whenever resources are busy. When resources are unavailable, implementations are able to either queue the traffic or take other action appropriate to the type of traffic.
- Implementations (except end-only, closed PB ones) either provide or are provided access to a user directory. This implies a tabular directory in which any DGN.DEN can be associated with any RGN.REN or error indicator. It also implies that DGN.* or *.* can be associated with any RGN.REN or error indicator.
- Re-direction Responsibilities

  Base Implementations (except end-only, closed PB ones):

- Change the destination DSU for any user destination found not to be local to the re-directing DSU.
- Do not change the destination DSU for any node-destinations. A node-destination is identified by a destination list entry that contains no users.
- Support the Intervention List. When a distribution is received containing any DSU name that matches an entry in the intervention list, the directing process is invoked.

Base Operations

Implementations support the following connection control verbs:

- Start_Connection
- Reset_MU_ID_Registry
- List_Control_MU_Queue
- Terminate_Connection

Implementations provide the operator with the functions defined for the following distribution control verbs:

- List_Queues_Containing_Distribution
- List_Queue_Entries
- Get_Distribution_Info
- Purge_Queue_Entry
- Hold_Distribution_Copy
- Release_Distribution_Copy

Implementations also support the following operations verbs that list, add, and remove DSU operations information:

- List_DSU_Data
- Add_DSU_Data
- Remove_DSU_Data
- Modify_DSU_Data

for the following DSU data structures:

- Directory
- Routing table
- Intervention list
- DSU definition
- Connection definitions
- Next-DSU queue definitions
- Agent list
- Server list
- MU_ID registry

Implementations log exception condition reports in an exception log and support the Get_Exception_Log_Entry verb to display the logged information.

Base Receive-time Checks

DSUs identify distributions for which they must accept responsibility at the PB by the high-integrity parameter and from other DSUs by the presence of the MU_ID. Before a DSU accepts responsibility for a distribution, it checks that it either can successfully process it or, if unsuccessful, can report the failure.

- The encoding structure specifying the types and levels of service required must be parsed and compared with the capabilities of the receiving DSU to ensure that it is capable of successfully processing the request.
- The encoding structure specifying the reporting requested must be parsed and examined.
- If reporting is requested, the encoding structures comprising the distribution identification, agent correl, and report-to information must be parsed and checked for consistency with lengths, contents, and conditions of presence, as defined in Appendix G.

Base Up-level Co-existence Capabilities

In order to coexist gracefully in future networks containing DSUs with additional, yet-to-be-imagined function, current implementations must provide certain toleration features.

- Implementations must gracefully reject any MU types they do not recognize.
- Implementations of certain electives, such as byte-count restart, must gracefully accept rejections of their elective protocols.
- Implementations of certain option sets, such as FS1 support, must accept, adjust to, and remember rejections received from adjacent DSUs.
- Implementations must, within limits, gracefully pass through unrecognized structures when they are present within a parent defined as capable of containing unrecognized children. These structures must be passed in encodings that are transferred to adjacent DSUs. The limits of number of unrecognized children and their total lengths are defined in Appendix F and Appendix G.
- Implementations must ignore unrecognized flags in the distribution flag bytes that are not mandatory for the particular role and reject distributions that have unrecognized flags in the bytes that are mandatory for the particular role.

Enhanced Character Strings Option Set

Implementations designed to share a network with FS1-only DSUs, whether or not they plan to connect directly to them with the FS1 Support Option Set, may choose this option set to ensure any-to-any connectivity. Newly assigned user names do not contain any characters outside CGCSGID 01134-00500.

Implementations supporting this option set have the ability to send, receive, and contain in their internal tables user names comprised of the character set, and observing the string conventions, specified in Appendix G.

For closed protocol boundary implementations in networks that contain terminal devices that cannot support the full character set, the network administrator constrains the choice of characters to those that are common to all terminals in the network to allow any-user-to-any-user communication.

Implementations of this option set also tolerate the enhanced character strings in DSU names they receive, routing the MUs and forwarding the DSU names unchanged. Except for products with special migration requirements, however, DSU names do not use the enhanced character string convention. Newly assigned DSU names do not contain characters outside CGCSGID 01134-00500.

Format Set 1 Support Option Set

- An implementation supporting this set is able to build MUs in either format set. If an adjacent DSU is only FS1-capable, this implementation will build FS1 MUs when sending to it, and will parse the FS1 MUs received from it.
- Implementations of this option set support four DS transport service TPs:
  - FS2 DS_Send
  - FS2 DS_Receive
  - FS1 DS_Send
  - FS1 DS_Receive
- Implementations of this option set can receive a distribution in an FS1 MU and forward it in an FS2 MU. Any and all function expressible in an FS1 MUs can also be conveyed in FS2. (In some reporting cases, one FS1 DRMU will be split into multiple FS2 DRMUs.)
- Implementations of this option set can receive certain FS2 MUs and forward them as FS1 MUs. Not all the function expressible in an FS2 MU can be expressed in an FS1 MU. The restrictions that apply to distributions that must be able to flow through FS1-only DSUs are described in Appendix D.
- Implementations of this option set support the *capacity* service parameter REQUIRE_SUPPORT_FOR X'FF' in FS1 MUs. This means they are able to receive distributions of any size until their storage resources are overrun. FS2 capacity service parameters of REQUIRE_LEVEL_GE 16MB (OR 1MB OR 4MB) are converted to REQUIRE_SUPPORT_FOR INDEF in FS1 and vice versa.

Security Option Set

Implementations of this option set support distributions that require security, as well as those that do not, in DS networks containing a mix of trusted and untrusted DSUs.

Distributions with a *security* service parameter of REQUIRE_LEVEL_GE LEVEL2 can be received and responsibly routed.

Operator Rerouting Option Set

This set consists of the function implied by the operations verb Reroute_Distribution_Copies.

Enhanced Connection Operations Option Set

Implementations of this set support the following verbs:

- List_Conversations
- Terminate_Conversation
- List_Distributions_Being_Sent
- List_Distributions_Being_Received
- List_Adjacent_DSUs
- List_Connections

Distribution Logging Option Set

Implementations supporting this set log the distributions that pass through the DSU. Each distribution copy is accounted for, either by an entry that consolidates the recording of each copy or by individual MU logs. When distributions are logged for cases other than exception conditions, the amount of information logged includes whatever is needed to support the Get_Distribution_Log_Entry. Implementations supporting this set will typically provide the installation with some means of limiting the amount of logging performed.

Electives

Certain functions may be implemented in more than one way. In some cases the different implementations will result in differences that can be perceived outside the DSU. If other implementations, either of DS or of the architectures that use it, must make any effort to cope with these differences, then that choice is defined in the architecture as an *elective*.

Electives have relatively minor effects on partner DSUs. All implementations can cope with whichever choices their partners make. If an implementation's choice differs from its partner's choice, its responsibility is limited to tolerating the effect of its partner's choice. Typically, this means gracefully ignoring certain actions the partner DSU takes. If it has made the same choice, its responsibility will usually be greater, and the two DSUs may achieve much better performance as a result.

Electives are not optional functions. Optional functions are defined in option sets. Electives are choices as to how or when a function is provided. Implementations make elective choices for performance or development cost reasons.

All electives are documented in the architecture because they, at least potentially, affect connectivity. The list of DS electives is as follows.

Electives within Base Function

1. Folding data priorities. Implementations may elect to fold the 16 DATA priorities into 2 groups: 1 through 8, and 9 through 16. The 1 through 8 group is called DATALO. When DATALO is used at the PB as a priority service level, it is the equivalent of DATA_4. Similarly, 9 through 16 are called DATAHI, the equivalent of DATA_12. Within each group, the distributions are sent on a first-in, first-out basis.

2. Receive-time enhancements. Implementations may elect to perform syntax checking, routing, and directing functions at receive time and, when appropriate, report exception conditions to the sending DSU by a REMU. Any exception conditions so detected and reported must apply to the entire distribution, not to only some of its destinations. Since the sending DSU still has responsibility for the distribution, it performs whatever DS reporting is required.

3. Next-DSU queue scheduling. Implementations may elect not to initiate sending as soon as a distribution is placed in a next-DSU queue. For example, time-of-day or queue depth are criteria that could inhibit the initiation of sending. Any such inhibiting controls can be removed by operator action.

4. Protocol electives. These electives do not apply to the protocols used in the FS1 option set.

   a. Single-session. Implementations may elect to restrict their connections to a single session: that is, they may elect not to provide support for parallel sessions on that connection. This means that inbound and outbound conversations must serially share the single session. This choice is made to reduce development effort at the cost of performance. The MU_ID management required for high-integrity traffic is considerably reduced by the single-session restriction, since only one MU_ID is in progress at a time.

   b. Receiver-limited conversation. After receiving the first DTMU on a conversation, the receiver can control how many more, if any, it will accept on that conversation. After sending the first DTMU, the sender turns the conversation around, allowing the receiver to indicate that the conversation should be ended by sending a CRMU with the *terminate_conversation* flag turned on. This leaves the conversation up, so that any outstanding control MUs can be exchanged, after which the sender deallocates. If no high-integrity traffic has been received on the connection, and therefore no control MUs are being exchanged, the receiver may deallocate the conversation directly.

   c. Sender-limited conversation. After the first DMU, if the receiver has not specified that the conversation be terminated, the sender has the choice of sending another DMU after the PRMU. If the receiver continues to accept additional DMUs indefinitely, it becomes the sender's responsibility to determine when to end the conversation. Typically, it sends until the queue is empty. This is usually the best choice, since starting conversations can be time and resource consuming. When the sender does decide to end the conversation, it sends any pending control MUs and deallocates the conversation.

   d. Receiver byte-count restart

      Implementations of this elective support the byte-count parameters of the mid-MU restart CRMU and interact with the storing server with the following verbs:

      - Initiate_Write with the *restartability*, *restart_ID*, and *restart_byte* parameters
      - Query_Last_Byte_Received

- Terminate_Restartability

  Sending implementations that do not support byte-count restart ignore the receiver-supplied byte-count and restart at the LLID.

e. Sender byte-count restart

   Implementations of this elective support the byte-count parameters of the MU restart protocols and interact with the fetching server with the following verbs.

   - Initiate_Read with the *restartability*, *restart_ID*, and *restart_byte* parameters
   - Terminate_Restartability

f. Sending SEMUs for basic-integrity traffic

   When a sender detects an exception condition and aborts the transfer of a DMU with a Send_Error, it may or may not choose to follow-up with a SEMU. If it does, the SEMU will not contain an MU_ID. Receivers must tolerate either choice.

5. Multiple local-delivery queues. Implementations may elect to have multiple local-delivery queues for each user. For example, each combination of user and agent can have its own queue. Within a user/agent combination, queues could be further differentiated by service parameters, for example, by priority.

6. Selective local-delivery. Usually, the local-delivery queues will be set up to accept any service parameters; however, the architecture allows the distribution to be rejected because there is no appropriate local-delivery queue for a particular service parameter. For example, a distribution specifying *security*: REQUIRE_LEVEL_GE LEVEL2 might not be deliverable to some of its user destinations because those users were not cleared for LEVEL2 and would not have LEVEL2 local delivery queues.

7. Informing sender when receiving server terminates before end-of-object. A specific server can return object acceptance before the complete server object has been delivered. DS_Receive can then either continue to receive, and discard the server object, or it can send a REMU to DS_Send with a *retry_action* of X'03'. The mid-MU restart CQMU/CRMU exchange will cause the distribution to be continued following the server object.

Electives within the Format Set 1 Support Option Set

1. LU 6.2 Deallocate (Type = SYNC_LEVEL). FS1 implementations may elect to send a separate Confirm flow or combine it with the Deallocate.

Specializations

Depending upon their role and PB choices, implementations can specialize, both in the network configuration they support and the types of traffic they can handle. The following table illustrates how those choices interact to provide greater or lesser potential for specialization.

Role	A) Closed PB	B) Open PB
1) Origin-only	Specialization can very greatly reduce required function	This combination of choices is precluded architecturally
2) Destination-only	Specialization can very greatly reduce required function	This combination of choices is precluded architecturally
3) Origin and Destination (End-only)	Specialization can greatly reduce required function	Configuration specialization can reduce required function
4) Intermediate-only (no local applications)	Application interface is eliminated by definition	This combination of choices is precluded by definition
5) All Roles	Traffic specialization can reduce application interface requirements	No specialization--base (and chosen option sets) must be fully implemented

Figure 55. Effect of PB and Role Choices on Specialization Potential

Choosing not to provide an open PB (column A in the table) has the greatest effect on specialization potential. It means that the originating and delivery interfaces with the applications can be highly specialized, and, for some roles, completely omitted.

Choice of role further affects specialization potential. The following specializations are identified by the cells of the table (1A, 3B, etc.) to which they apply.

- No through traffic (1A, 2A, and 3A)

  DSUs that do not have to pass traffic between one connection and another--that is, the intermediate role--or between an open PB and the DS network, can specialize in only the types of traffic and services needed by their private applications. Properly set up routing tables and correctly specified distribution requests ensure that such DSUs will never be sent anything outside their speciality. If such a DSU were to receive inappropriate traffic, it would reject it, probably at receive time. The following specializations derive from the no-through-traffic situation.

  — Single-user traffic (2A, and 3A)

    DSUs that are so configured that they never can have more than one local user will never be called upon to perform destination fan-out.

  — No user traffic (1A, 2A, and 3A)

DSUs that are specialized for applications that never specify user destinations will never have to make reference to a user directory, and neither generate nor accept distributions with user names in them.

  - Specialization by service parameters

    Closed PB, end-only DSUs that are specialized in a limited set of service types and levels will never be called upon to originate or deliver distributions requesting any other services.

- Closed PB specializations (1A, 2A, 3A, and 5A)

  1. Specialization by server (1A, 2A, 3A, and 5A)

     Closed PB DSUs that are specialized in distributions for only one type of specific server will never be called upon to deliver distributions to another type of specific server or to the general server.

  2. Distributions with no specific server name (1A, 2A, 3A, and 5A)

     Closed PB, destination DSUs that are specialized in distributions for only the general server will never be called upon to deliver distributions to any type of specific server.

  3. Distributions with no server objects (1A, 2A, 3A, and 5A)

  4. Specialization by agent (1A, 2A, 3A, and 5A)

     Closed PB DSUs that are specialized in distributions for a particular agent will never be called upon to deliver distributions to any other agent.

- No routing needed

  - No out-going traffic (2A)

    Destination-only DSUs are never called upon to make outgoing routing decisions and therefore do not need to support a routing table.

  - Single connection configuration (1A, 3A, and 3B)

    End-only DSUs that are always so configured that they have only one connection will never be called upon to make any route selection, and therefore do not need to support a routing table.

Optimizations

Optimizations can be either perceptible outside the DSU or not. If they are not, then, by definition, they cannot be of interest to architecture or other implementations.

If the effects of the optimization are perceptible outside the DSU but other implementations need make no effort to cope with those effects, then that choice is categorized as an optimization. This contrasts with electives, which do impose efforts on other implementations.

Examples of perceptible optimizations occur most frequently in exception processing and reporting. Two implementations could differ in the sequences of their checks and, as a result, might report the same condition with different report codes, or, when multiple exceptions were present, might detect and report different ones. A common partner could perceive the differences in the reports. Provided that the common partner did not have to make any special effort to handle either of the two reports, the choice would be categorized as an optimization.

Sometimes implementations will take different actions in certain exception conditions. For example, one implementation might deallocate a conversation immediately, where another would not. Since the partner has to be able to cope with a deallocation anyway, this difference would require no extra effort on its part.

This does not entitle implementations to be unreasonable. Field service requires report codes that are helpful in diagnosing a problem. Network operators require that conversations and sessions are not taken down arbitrarily.

In summary, optimizations are choices made by an implementation that do not affect the connectivity of implementations and are therefore not subject to precise architectural compliance rules. Therefore, they are not constrained by the architecture.

Appendix D. FS1/FS2 Coexistence

General Introduction to the Coexistence Strategy

The FS1/FS2 coexistence plan is based on a "translation" scheme, whereby a distribution encoded initially in one format set may be mapped to the other format set. For example, a user on an FS1-only product may send a distribution to a user on an FS2-only product. At some DSU along its route, the distribution is received in FS1 and forwarded on to the next hop in FS2. The information in the distribution is not changed, but its bit-level representation is redefined.

FS1 products are oriented toward DIA and S/3X Object Distribution agents. Implementations with no requirement to connect directly to FS1 products may choose to provide only FS2 support. Implementations with a requirement to connect directly to FS1 products provide both FS1 and FS2 support. That is, new DS implementations supporting DIA or S/3X Object Distribution applications with a migration requirement to communicate directly with the FS1 DIA and S/3X Object Distribution products will support FS1 as well as FS2. Other products need support only FS2.

Some new FS2-only DS implementations may support local DIA or S/3X Object Distribution agents. End users on these new implementations can communicate with end users on FS1-only implementations if an FS1- and FS2-supporting DSU is along the distribution's path, and if the distribution does not exploit any FS2-unique function (which cannot be encoded in FS1).

To summarize the coexistence plan, certain DSUs will support both FS1 and FS2. These DSUs are said to be *bilingual*. Bilingual DSUs act as gateways between FS1-only portions of the DS network and FS2-supporting portions, translating between the format sets as necessary. This appendix describes how the bilingual DSU determines whether translation is possible, when it is required, and how it is accomplished.

General Actions for Handling Format Set 1 and Format Set 2 Coexistence

An FS1-encoding DSU can send and receive only FS1 MUs. Similarly, an FS2-encoding DSU can send and receive only FS2 MUs. Bilingual DSUs accept both FS1 and FS2 MUs. The "coexistence" problem, therefore, can be reduced to a simple question: What does a bilingual DSU do, and how does it know to do it?

The "how" portion of the above question depends on the distribution being processed and the capabilities of the DSU to which the distribution is to be sent. This topic is discussed in "Determining Partner's Encoding Level" on page 385.

The "what" portion of the question is discussed in "Detailed Actions for FS1/FS2 Coexistence" on page 386. The basic approach is a translation scheme, but using Format Set 2 whenever possible. Format Set 1 MUs are translated to Format Set 2 whenever possible, and Format Set 2 MUs are translated to

Format Set 1 as required. For example, if a bilingual DSU receives an FS1 Dist_MU *type* TRANSPORT and sends the distribution to another bilingual DSU, it uses Format Set 2 flows.

Coexistence Plan Constraints

All restrictions of the coexistence plan are related directly to one or more of the following points:

1. FS2 supports function that cannot be supported in FS1.
2. Most (if not all) new applications are expected to exploit some FS2-only function.
3. Current FS1 implementations are application-specific, suoporting DIA and S/3X Object Distribution applications.
4. New DS implementations are expected to support FS2.

Existing Functions

The coexistence plan is tailored to the use made of FS1 implementations by DIA and S/3X Object Distribution applications. Certain architectural features of FS1 that have not been used by DIA or S/3X Object Distribution agents (e.g., FS1 application-report distributions for applications other than DIA) are not supported by the coexistence plan. New DIA and S/3X Object Distribution agents may exploit FS2-only function, but these new features cannot be translated into FS1 and will typically cause distributions exploiting them to be terminated should they attempt to enter an FS1 network.

Topology

Since FS1 cannot support all of the function in FS2, new applications cannot, in general, communicate over a route containing any FS1-only DSUs. This, combined with the orientation of the coexistence plan to DIA and S/3X Object Distribution leads to the following topologies being supported:

- FS1 to FS2
- FS2 to FS1
- FS1 through FS2 to FS1

Since FS2 supplies more function than FS1, and since new applications will typically exploit this increased function, FS2-to-FS2 traffic must, in general, flow over an all-FS2 route. Occasionally, however, a system administrator may wish merely to continue using the FS1 applications, and not to use any new applications. For such system administrators, the coexistence plan supports the following topology:

- FS2 through FS1 to FS2 for existing (FS1) applications using function available only in FS1

  Unpredictable results may occur if system administrators configure their networks so that new FS2 applications attempt to communicate using an FS1 DSU as an intermediate DSU. Usually, distributions exploiting FS2-only features cannot be sent through FS1 intermediate DSUs, and will be rejected by the bilingual DSU attempting the translation.

End-User to End-User Connectivlty

This coexistence plan allows full end-user-to-end-user connectivity for existing FS1 applications. An end user supported by an FS1-only, FS2-only, or bilingual DSU can communicate with any other end user, without knowledge of the encoding-support level of the destination (or any other nonlocal) DSU. The encoding-support level of the origin DSU is not affected by (and does not affect) the encoding-support level of the destination DSU.

DSU-to-DSU Connectivity

A Format Set 1 DSU may connect directly to a Format Set 1 DSU or a bilingual DSU. A Format Set 2 DSU may connect directly to a Format Set 2 DSU or a bilingual DSU. A bilingual DSU may connect directly to a Format Set 1, Format Set 2, or bilingual DSU.

Determining Partner's Encoding Level

Each bilingual DSU must know the encoding-level capabilities of its logically adjacent DSUs (i.e., any DSU that can be a "next hop"). This knowledge is kept as a *data_stream_format* field for each next hop, and has a value of FORMAT SET 1, FORMAT SET 2 or ERROR.

The transaction program names for FS2 DS_Send and DS_Receive are different from those for FS1 DS_Send and DS_Receive. (See "Transaction Program and Server Names" on page 631 for the registered values.) A bilingual DSU takes the following steps to discover the encoding-support level of each of its logically adjacent DSUs. Discovering that the partner is an FS1-only DSU occurs prior to the FS2 MU_ID Registry synchronization process, which initializes the DS connection (see Chapter 2 for details of this synchronization process).

1. The default value for the *data_stream_format* field is FORMAT SET 2.
2. If the value of the *data_stream_format* field is FORMAT SET 2, a bilingual DSU issues an Allocate verb using the Format Set 2 TP names for DS_Send and DS_Receive.
3. If a verb after the Allocate fails because the attached TP name does not exist (*return_code* = ALLOCATION ERROR, TP NAME NOT RECOGNIZED), the bilingual DSU resets the *data_stream_format* field for the partner DSU to FORMAT SET 1 and retries the Allocate (see step 4).
4. If the value of the *data_stream_format* field is FORMAT SET 1, a bilingual DSU issues an Allocate verb using the Format Set 1 TP names for DS_Send and DS_Receive.
5. If a verb after the Allocate fails because the attached TP name does not exist (*return code* = ALLOCATION ERROR, TP NAME NOT RECOGNIZED), the bilingual DSU resets the *data_stream_format* field to ERROR and notifies the operator.
6. All FORMAT SET 1 values for the *data_stream_format* variables are periodically reset to FORMAT SET 2. The purpose of resetting these variables is to discover whether any logically adjacent DSUs have been upgraded to be bilingual. The exact mechanism for resetting these FORMAT SET 1 values is left to the discretion of the implementation. A suggested method is to automatically reset the values once a month (or some other appropriate period). Other methods include having the system operator reset the values period-

ically, or having the implementation always attempt to allocate a conversation using the Format Set 2 TP name (i.e., on each conversation).

7. If a bilingual DSU is attached by a Format Set 2 TP name, it resets its own *data_stream_format* field for the attaching DSU to FORMAT SET 2.

8. If a bilingual DSU is attached by a Format Set 1 TP name, it *does not* reset its own *data_stream_format* field for the attaching DSU to FORMAT SET 1. (Simply stated, the FS2 encodings are preferred. News that the partner DSU is capable of sending FS2 (see step 7) is sufficient reason to attempt to send FS2 to that partner; news that the partner DSU is capable of sending FS1 is *not* a sufficient reason to restrict communication to the partner to FS1.)

Detailed Actions for FS1/FS2 Coexistence

Figure 56 on page 388 summarizes the actions of a bilingual DSU for any given combination of inputs and next hops.

Inputs

In general, the protocol boundary and the distribution transport sublayer pass the same information into and out of the DSU. For example, a DSU may receive input from either a Send_Distribution verb or from DS_Receive. This does not mean that a single mechanism controls both the Send_Distribution verb and DS_Receive, nor does it imply that the same format is used across the agent protocol boundary and the LU 6.2 basic conversation protocol boundary. The important point is that the *information content* of the distribution is essentially the same, whether it enters the DSU via the Send_Distribution verb or via DS_Receive.

For this discussion, therefore, no distinction is made between data received across the agent protocol boundary and data received from a partner DSU. Similarly, the data carried out of the DSU by the Receive_Distribution verb and the data carried out by the DS_Send process are considered identical.

The following are the possible inputs to a bilingual DSU:

- Constrained transport information (without DIA report information)

  Constrained transport information is whatever can be held in an FS1 Dist_MU *type* TRANSPORT. If a distribution is encoded in FS2, no non-FS1 features may be exploited:

  — The origin and all destinations of the distribution must be users; the *report-to* destination, if specified, must also be a user.
  — The *agent_object* must be less than 512 bytes.
  — The *dest_agent*, if specified, must be DIA or DIASTATUS.
  — Neither the *supplemental_dist_info1* field nor the *supplemental_dist_info2* field may be specified.
  — The *seqno* field must have a value between 1 and 9999 (inclusive).

- Constrained FS2 distribution report information

  This contains only DS report information. Constrained FS2 distribution report information is whatever can be held in an FS1 Dist_MU *type* REPORT containing only DS report information. If a distribution is encoded in FS2, no non-FS1 features may be exploited:

  - The *reported-on* destinations and the *report-to* destination must be users.
  - A *reported-on_dest_agent* other than DIASTATUS may not be specified.
  - Neither the *reported-on_supp_dist_info1* nor the *reported-on_supp_dist_info2* field may be specified.

- Unconstrained FS2 information

  These distributions exploit one or more FS2-only features and therefore cannot be encoded in FS1. Attempting to send an unconstrained distribution to an FS1 DSU results in the distribution being terminated with an exception code of "Function conflicts with FS1 encodings" (X'1003 001C').

- DIA report

  In FS1, this information is encoded as a Dist_MU *type* REPORT with only DIA report information. If any DS report information were included in the DMU, the application (DIA) report information would be discarded. If an FS1 Dist_MU *type* REPORT contains only application report information (i.e., no DS report information) and the application is not DIA, the distribution cannot be translated into FS2 and is therefore terminated.

  Unlike FS1, FS2 treats agent-to-agent exception flows no differently from any other agent-to-agent exchanges, and DIA report information is encoded as a constrained DTMU with *origin_agent* set to DIA and *dest_agent* set to DIASTATUS.

- FS1 DS report

  This is an FS1 Dist_MU *type* REPORT with DS report information. If application (DIA) report information is also present in the Dist_MU, it is discarded.

Next Hop

The next hop of a bilingual DSU is either an FS1 DSU or an FS2 DSU. Because the default action is to use FS2 encodings, adjacent DSUs that are bilingual appear to be FS2-only DSUs.

Actions

The referenced notes refer to the list below the table.

	Actions of Bilingual DSU where	
INPUT	Next Hop is FS1	Next Hop is FS2
Constrained Transport Information	Send distribution in FS1 (See Note 1)	Send distribution in FS2 (See Note 5)
Constrained FS2 DRMU	(See Note 2)	Send distribution in FS2
Unconstrained FS2 Information	Terminate distribution (See Note 3)	Send distribution in FS2
DIA Report Information	(See Note 4)	(See Note 6)
FS1 DS Report	Send distribution in FS1	(See Note 7)

Figure 56. Summary of Detailed Actions of a Bilingual DSU

Notes:

1. The DSU builds an FS1 Dist_MU *type* TRANSPORT. If an FS2 DTMU was received, then see section "Transport Mapping" on page 389 for the structure-by-structure mapping.

2. The DSU builds a single FS1 Dist_MU *type* REPORT with *gen_SNADS_report*. See section "DS Report Mapping" on page 391 for further detail.

3. *Any* unconstrained information in a distribution causes the entire distribution copy to be terminated. For example, if a DTMU with multiple destinations is received and one of the destinations is a node, rather than a user, the entire distribution copy is terminated for *all* destinations—user and node destinations alike.

4. If an FS1 Dist_MU *type* REPORT with DIA report information is received, the DSU simply forwards the DMU on to the next hop.

   If a constrained FS2 DTMU is received, DIA report information is present when the destination agent is DIASTATUS. The DSU builds an FS1 Dist_MU *type* REPORT with a DIA report. The DIA report portion of the FS1 *dist_command* is taken via simple byte-by-byte copy from the *server_object* of the input FS2 MU. The FS1 MU does not contain a *server_object*. See section "DIA Report Mapping" on page 395 for further detail.

5. The DSU builds an FS2 DTMU. If an FS1 Dist_MU was received, then see section "Transport Mapping" on page 389 for the structure-by-structure mapping.

6. If an FS2 DTMU is received, the DSU simply forwards the DMU on to the next hop, handling this case exactly as it would any other FS2 MU, regardless of the agent specified.

   If an FS1 Dist_MU *type* REPORT with DIA report information is received, the DSU builds an FS2 DTMU specifying the *dest_agent* as DIASTATUS. The portion of the FS1 command containing the DIA report is inserted via simple byte-by-byte copy into the *server_object*. See section "DIA Report Mapping" on page 395 for further detail.

7. The DSU generates one to four FS2 DRMUs from the FS1 Dist_MU *type* REPORT with DS report (and discards any DIA report information). See section "DS Report Mapping" on page 391 for further detail.

Placement of Conversion Actions

The placement of the FS1/FS2 mapping functions described below depends on the implementation. However, this appendix is written assuming that all mapping functions will be performed in the FS1 distribution transport sublayer (i.e. the FS1 DS_Send and DS_Receive processes). There is no reason to treat *any* distribution that is received in FS2 and forwarded on to the next hop in FS2 differently from any other FS2-only distribution.

Transport Mapping

All distributions received as FS1 MUs can be built in FS2. Some distributions received as FS2 DTMUs, however, may exploit function that cannot be contained in FS1. Such distributions are termed "unconstrained"; if an attempt is made to send them to an FS1-only DSU, they are terminated by the bilingual DSU with an *SNA_report_code* of "Function conflicts with FS1 encodings" (X'1003 001C'). Constrained distributions can be mapped from FS2 to FS1.

This section gives the structure-by-structure mapping between Format Set 1 and Format Set 2 for Transport MUs. Since the supported mappings are reversible, Figure 57 suffices for both the FS1 to FS2 and the FS2 to FS1 mappings.

Transport Atomic Structure Mappings	
FS2 Structure	FS1 Structure
HOP_COUNT	HOP_COUNT
DIST_FLAGS	DIST_FLAGS
SERVICE_PARMS	SERVICE_PARMS
SERVER_OBJ_BYTE_COUNT	SERVER_OBJ_BYTE_COUNT
ORIGIN_AGENT	DEST_AGENT
SERVER	SERVER
ORIGIN_RGN	ORIGIN_RGN
ORIGIN_REN	ORIGIN_REN
ORIGIN_DGN	ORIGIN_DGN
ORIGIN_DEN	ORIGIN_DEN
SEQNO_DTM	ORIGIN_SEQNO, ORIGIN_DTM
AGENT_CORREL	AGENT_CORREL
REPORT-TO_RGN	REPORT-TO_RGN
REPORT-TO_REN	REPORT-TO_REN
REPORT-TO_DGN	REPORT-TO_DGN
REPORT-TO_DEN	REPORT-TO_DEN
REPORT_SERVICE_PARMS	REPORT_SERVICE_PARMS
REPORT-TO_AGENT	REPORT-TO_AGENT
DEST_RGN	DEST_RGN
DEST_REN	DEST_REN
DEST_DGN	DEST_DGN
DEST_DEN	DEST_DEN
AGENT_OBJECT	AGENT_OBJECT
SERVER_OBJECT	SERVER_OBJECT

Figure 57. FS2 to FS1 Mapping for Transport MUs

Notes on execution-time checks and actions, FS2-to-FS1:

1. Unless otherwise noted, the FS2 structure values are simply copied to their FS1 counterparts.
2. An FS2 DTMU with third-party reporting might specify a *report-to_user* with no *report-to_DSU.* In this case, the bilingual DSU uses its own DSU name as the *report-to_DSU* for the Format Set 1 MU.
3. The Format Set 2 *agent_correl* field will be truncated to 44 bytes, if necessary.
4. Unrecognized fields in the command or at the highest level are ignored (and discarded).
5. The *exception_report* flag of an FS2 DTMU is mapped to the correspondingly-named counterpart flag in the FS1 MU. The FS1 *distribution_type* flag (i.e., TRANSPORT or REPORT) is set based on the *dest_agent* of the Format Set 2 MU; see section "DIA Report Mapping" on page 395 for further details. Unused bits in the Format Set 2 *dist_flags* byte 4 are ignored (whatever their value). The Format Set 2 *dist_flags* byte 3, bits 0-7, and *dist_flags* byte 2, bits 1-7 must be 0; otherwise, the distribution is terminated.
6. The FS1 *protection* service parameter is set to YES.
7. If the FS2 *capacity* service parameter is REQUIRE_LEVEL_GE ZERO or the FS2 *priority* is greater than DATA16, then the FS1 *capacity* is ZERO. Otherwise, the FS1 *capacity* is INDEFINITE.
8. If the date/time stamp of the DTMU contains GMT plus local offset, the FS1 MU is given the origin's "local" time. That is, the local-time offset is added to the GMT time to yield the date/time stamp of the FS1 MU.
9. If a *dest_agent* of DIA or DIASTATUS is specified, the FS1 agent is DIA.

Notes on execution-time checks and actions, FS1-to-FS2:

1. Unless otherwise noted, the FS1 structure values are simply copied to their FS2 counterparts.
2. If the FS1 distribution specifies the *capacity* service parameter as INDEFINITE, the FS2 distribution *capacity* is set to REQUIRE_LEVEL_GE 16MB.
3. The *server_object_byte_count* field, if supplied in the Format Set 1 MU, is assumed to be correct and copied to the Format Set 2 MU. No checking is performed.
4. Bits 3-7 of the FS1 *dist_flags* must be 0, or the distribution is terminated. Bit 2 is ignored, whatever its value. The *exception_report* flag of a Format Set 1 DMU is mapped to the correspondingly named counterpart flag in the Format Set 2 MU. The Format Set 1 *distribution_type* flag (i.e., TRANSPORT or REPORT) is not carried over directly into the Format Set 2 MU; see section "DIA Report Mapping" on page 395 for further details. All unassigned flags in the Format Set 2 MU are set to 0.
5. The *server_parms* field is not supported in FS2. If it is present in the FS1 MU, the distribution is terminated.

6. The date/time stamp of the FS1 MU is used as the date/time stamp for the FS2 MU, and is considered "local" time only (that is, not GMT or GMT-plus-offset).
7. The FS2 distribution is sent with *high* integrity.

DS Report Mapping

This section discusses in detail the Format-Set-1/Format-Set-2 mappings for DS reports.

All distributions received in FS1 can be built in FS2. Distributions received in FS2, however, may exploit function that cannot be contained in FS1. In such cases, the DSU terminates the distribution and logs the error with the *SNA_report_code* of "Function conflicts with FS1 encodings" (X'1003 001C').

FS2 reports carry both origin and report-to agent information, if both are specified. FS1 reports carry only one agent name, and that agent must reflect the report-to agent, if specified. Therefore, if a distribution

1. originates in an FS2 subnet
2. and exploits the *report-to_agent* parameter
3. and has a distribution report that is generated in or passes through an FS1 subnet

the report will indicate that the distribution was originated by the *report-to_agent* instead of the true originating agent.

Also, if a distribution specifies a *report-to_user*, then the *reported-on_origin_DSU* field will be absent from the FS2 DRMU.

The mapping between the Format Set 1 Dist_MU *type* REPORT and the FS2 DRMU is given in Figure 58 on page 392.

DS Report Atomic Structure Mappings	
FS2 Structure	FS1 Structure
HOP_COUNT	HOP_COUNT
SERVICE_PARMS	SERVICE_PARMS
REPORT-TO_AGENT	ORIGIN_AGENT
REPORTING_RGN	ORIGIN_RGN
REPORTING_REN	ORIGIN_REN
REPORT_DTM	ORIGIN_DTM
REPORT-TO_RGN	DEST_RGN
REPORT-TO_REN	DEST_REN
REPORT-TO_DGN	DEST_DGN
REPORT-TO_DEN	DEST_DEN
REPORTED-ON_ORIGIN_DGN	REPORTED-ON_ORIGIN_DGN
REPORTED-ON_ORIGIN_DEN	REPORTED-ON_ORIGIN_DEN
REPORTED-ON_SEQNO_DTM	REPORTED-ON_SEQNO
REPORTED-ON_SEQNO_DTM	REPORTED-ON_DTM
REPORTED-ON_AGENT_CORREL	REPORTED-ON_AGENT_CORR
REPORTED-ON_DEST_DGN	REPORTED-ON_DEST_DGN
REPORTED-ON_DEST_DEN	REPORTED-ON_DEST_DEN
RECEIVING_RGN	DETECTING_RGN
RECEIVING_REN	DETECTING_REN
SNA_REPORT_CODE	GEN_SNADS_COND_CODE

Figure 58. FS1 to FS2 DS Report Mapping

Notes on execution-time checks and actions, FS2-to-FS1:

1. Unless otherwise noted, the FS2 structure values are simply copied to their FS1 counterparts.
2. The destination of the DRMU must be a user rather than a node. If the FS2 DRMU specifies the *report-to_DSU_user* as a node, the distribution is terminated.
3. Unrecognized fields in the *report_command* or *report_information* or at the highest level are discarded.
4. The presence of either the *reported-on_supp_dist_info1* or the *reported-on_supp_dist_info2* field causes the distribution to be terminated.
5. The *reported-on_origin_RGN* and the *reported-on_origin_REN* parameters are discarded. (The *reported-on_origin_DGN* and *reported-on_origin_DEN* must be present.)
6. The *origin_seqno* parameter is set to X'F0F0F0F0,' as specified in Appendix G.
7. The *dist_flags* are set to indicate *type* (REPORT), *exception_report* (NO).
8. The *SNA_report_code* is mapped to a DS condition code. See Appendix E for the exact mappings.
9. The *reported-on_dest_DSU* structure is ignored.
10. The *reported-on_agent_correl* structure is truncated to 44 bytes, if necessary.
11. If the *reported-on_seqno* is greater than 9999, the distribution is terminated.
12. The *protection* service parameter is set to YES, and the *capacity* is set to ZERO.

13. The *reported-on_origin_agent*, if present, is ignored.

Notes on execution-time checks and actions, FS1-to-FS2:

1. Unless otherwise noted, the FS1 structure values are simply copied to their FS2 counterparts.
2. The *origin_user* field in the Format Set 1 Dist_MU *type* REPORT should not be present. If present, it is discarded; only the *origin_DSU* is carried in the FS2 MU.
3. Exception reporting is handled differently in FS1 and FS2. FS1 allows a different DS condition code to be reported for each destination, but provides minimal diagnostic report information. In FS2, a single MU reports only one exception, but extensive reporting information may be provided.

   If a bilingual DSU receives an FS1 MU with multiple condition codes, it generates multiple FS2 DRMUs, one DRMU for each unique condition code. This can be done by sorting the FS1 condition codes, and generating an FS2 DRMU for each unique code. A single FS1 Dist_MU *type* REPORT with a DS Report may cause a bilingual DSU to generate up to four FS2 DRMUs. For more detail of this process, refer to "FS1 Specific DS Reports."
4. The DS condition code is mapped to an *SNA_report_code*. See section Appendix E. for the exact mappings.
5. The *origin_seqno* field is ignored (and discarded).
6. The *dist_flags* are set to specify *exception_report* (NO).
7. The presence of a *server_object* or *agent_object* in the FS1 MU causes the distribution to be terminated.
8. All reported-on destinations are users. However, the *reported-on_dest_RGN* and *reported-on_dest_REN* are omitted within the *reported-on_dest_DSU*.
9. The FS2 distribution is sent with *high* integrity.

FS1 Specific DS Reports

As mentioned earlier, a single FS1 Dist_MU *type* REPORT with a DS report may contain different condition codes. In practice, multiple exceptions rarely occur simultaneously for a single distribution, so typically the report contains only one condition code. Nonetheless, to understand how many unique condition codes may occur in a single FS1 DS report, divide the condition codes into three groups (see Figure 59 on page 394.) The first group contains only Routing Exception and Unknown User Name, which are the FS1 exception conditions having a scope of "Destinations" as described in Appendix E. If an FS1 DSU receives a distribution and checks for these exceptions, some destinations might have a Routing Exception, some others might have an Unknown User Name, and others may have neither exception condition.

The second group of condition codes contains Unknown Resource Name—Specific Server, Invalid Server Parameters, Unknown Resource Name—Agent TP and Specific Server Exception. The scope of these exceptions is "Local-only Destinations." These exceptions can occur only at the destination DSU. Each of these conditions must apply to all of the local destinations or none of the local destinations. For example, if the specific server is unknown for one local user, it will be unknown for all local users.

The third group of condition codes includes all the other exception conditions, each having the scope of "Distribution Copy." Each of these conditions applies either to all destinations or no destinations of an MU. For example, Hop Count Exhausted is in this third group. Since the hop count is maintained on a distribution copy basis, the Hop Count Exhausted exception will affect all destinations in that distribution copy.

Given any distribution, an FS1 DSU may detect either or both codes from group 1, but at most one from group 2 and at most one from group 3. Thus, the FS1 DSU may report up to four condition codes for any given MU.

Codes that might apply to some destinations and not others:
Routing exception
Unknown user name

Codes that apply only at destination DSUs, and apply to all destinations or no destinations:
Unknown resource name—specific server
Invalid server parameters
Unknown resource name—agent TP
Specific server exception

Codes that must apply to all destinations or no destinations:
Hop count exhausted
Format exception
Function not supported
Operator intervention—purged
User names lost
Resource not available
System exception
Insufficient resource
Storage-medium exception
REMU exception
Server object size incompatible with capacity level

Figure 59. Groupings of DS FS1 Condition Codes

For example, suppose that a DSU with a DSU name of D1 receives a Dist_MU *type* TRANSPORT. Also, assume that the DSU names D2, D3, and D4 appear in the intervention list at D1. Further, suppose that the DMU's received hop count value was 1, that a format exception was detected during receipt, and that neither the agent nor the server specified in the DMU exists at this DSU. The destination list (and the errors associated with each destination) is given below:

- User=U1, DSU=D1. The directory indicates that U1 is an unknown user name.
- User=U2, DSU=D2. The directory indicates that U2 is an unknown user name.
- User=U3, DSU=D3. This is a valid local user.
- User=U4, DSU=D4. This is a valid local user.
- User=U5, DSU=D5. The routing table indicates that D5 is a routing error.
- User=U6, DSU=D6. The routing table indicates that D6 is a routing error.
- User=U7, DSU=D7. The directory indicates that U7 is an unknown user name. The routing table indicates that D7 is a routing error.

- User=U8, DSU=D8. The directory indicates that U8 is an unknown user name. The routing table indicates that D8 is a routing error.
- User=U9, DSU=D9. This is a valid remote user.

This DSU happens to check for exceptions in the order given in Figure 59 on page 394. Routing exceptions are checked first, and destinations U5, U6, U7 and U8 are flagged. Unknown user names are checked next, and U1 and U2 are flagged. Users U7 and U8 would have been flagged for unknown user names, but they were previously flagged as being routing exceptions. The check for Unknown Resource Name—Specific Server causes the remaining local users to be flagged, U3 and U4. Unknown Resource Name—Agent TP will not be reported, since all of the valid local users (U3 and U4) have already been flagged as having Unknown Resource Name—Specific Server. Since the received hop count is 1, the distribution cannot be forwarded because a Hop Count Exhausted condition exists. All of the remaining remote users (only U9 in this example) will thus be flagged with Hop Count Exhausted. Format Exception will not be reported because no valid destinations remain. Thus, in spite of the fact that six exception conditions exist, only four (Routing Exception, Unknown User Name, Unknown Resource name—Specific Server and Hop Count Exhausted) will be reported.

DIA Report Mapping

This section discusses in detail the Format-Set-1/Format-Set-2 mappings for DIA reports. This mapping is unusual in that an FS1 report flow is mapped to (or from) an FS2 transport flow. Figure 60 shows the atomic structures not directly related to the DIA report, and Figure 61 on page 397 shows how FS2 DTMUs contain an FS1-style DIA report in a mixed network.

A DSU supporting an open protocol boundary makes the DIASTATUS and DIA agent names available to the local DIA application program. This application program should send DIA report information using a Send_Distribution verb with *origin_agent* set to DIA, *dest_agent* set to DIASTATUS and *server* set to DIA.

FS2 DTMU to FS1 DIA Report Structure Mappings	
FS2 Structure	FS1 Structure
HOP_COUNT	HOP_COUNT
DIST_FLAGS	DIST_FLAGS
SERVICE_PARMS	SERVICE_PARMS
ORIGIN_AGENT/DEST_AGENT	DEST_AGENT
SERVER	SERVER
ORIGIN_RGN	ORIGIN_RGN
ORIGIN_REN	ORIGIN_REN
ORIGIN_DGN	ORIGIN_DGN
ORIGIN_DEN	ORIGIN_DEN
SEQNO_DTM	ORIGIN_SEQNO, ORIGIN_DTM
AGENT_CORREL	AGENT_CORREL
DEST_RGN	DEST_RGN
DEST_REN	DEST_REN
DEST_DGN	DEST_DGN
DEST_DEN	DEST_DEN

Figure 60. FS2 DTMU to FS1 Dist_MU type Report (DIA Report) Mapping

Notes on execution-time checks and actions, FS2-to-FS1:

1. Unless otherwise noted, the FS2 structure values are simply copied to their FS1 counterparts.
2. The fields of the FS1 Dist_MU *type* REPORT that are not specifically devoted to agents or report information (e.g., *origin_RGN* and *origin_REN* of the FS1 Dist_MU *type* REPORT), are mapped from FS2 DTMU's *prefix*, *command*, and *destination_list* portions according to the rules given above for transport MU mapping (see section "Transport Mapping" on page 389).
3. If the DTMU's *priority* service parameter is less than CONTROL, the FS1 MU's *priority* is set to CONTROL. FS1 *protection* is set to YES, *capacity* is set to ZERO.
4. The contiguous section of the FS1 Dist_MU *type* REPORT consisting of the *dist_report_operands* (see Figure 57 on page 389) is copied from the FS2 MU's *server_object*.
   - The byte stream to be copied to the FS1 Dist_MU *type* REPORT begins with the first LLID inside the *server_object*, which must be ID=C340 or ID=C361. (Otherwise, the translation is terminated.)
   - Subsequent LLIDs of the *server_object* are also copied into the FS1 Dist_MU *type* REPORT without modification, until the termination condition is detected.
   - The termination condition is two consecutive LLIDs with LL=5, ID=C351. The two terminating LLIDs are included in the FS1 Dist_MU *type* REPORT.
   - Exhaustion of the *server_object* without detecting the termination condition causes the translation (and distribution) to be terminated.
   - Any data left in the *server_object* is discarded.
5. The FS1 *dest_agent* is set to DIA. The *distribution_type* bit is set to REPORT.
6. No FS1 *server* is specified.
7. The *server_object_byte_count* is ignored (if supplied).

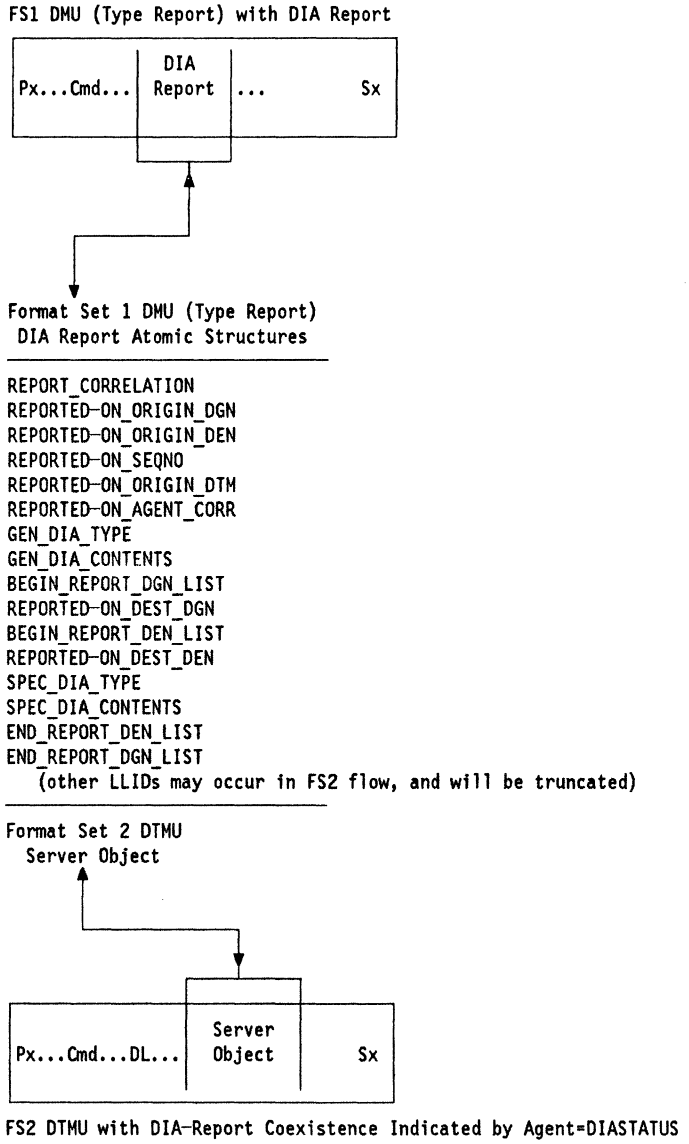
Notes on execution-time checks and actions, FS1-to-FS2:

1. Unless otherwise noted, the FS1 structure values are simply copied to their FS2 counterparts.
2. The fields of the FS2 DTMU *prefix*, *command*, and *destination_list* that are not specifically devoted to agents or report information are mapped to the FS1 Dist_MU *type* REPORT according to the rules given above for transport MU mapping (see section "Transport Mapping" on page 389).
3. If the FS1 *priority* is CONTROL, then the FS2 *priority* is set to REQUIRE_LEVEL_GE DATA16. FS2 *capacity* is set to REQUIRE_LEVEL_GE 16MB.
4. The FS1 Dist_MU structure *dist_report_operands* (see Figure 61 on page 397) is copied to the FS2 MU's *server_object*. (This means that the first ID of the *server_object* will be either C340 or C361; the last two IDs of the *server_object* will be identical: LL=5, ID=C351.)

5. The FS2 DTMU's *dest_agent* is DIASTATUS, the *origin_agent* is DIA and the Server is DIA. The *exception_report* flag is set to NO.

6. Bilingual DSUs may calculate the length of the *server_object* and supply the *server_object_byte_count*.

7. The FS2 distribution is sent with *high* integrity.



```
FS1 DMU (Type Report) with DIA Report

Px...Cmd... | DIA Report | ...   Sx

Format Set 1 DMU (Type Report)
 DIA Report Atomic Structures
______________________________
REPORT_CORRELATION
REPORTED-ON_ORIGIN_DGN
REPORTED-ON_ORIGIN_DEN
REPORTED-ON_SEQNO
REPORTED-ON_ORIGIN_DTM
REPORTED-ON_AGENT_CORR
GEN_DIA_TYPE
GEN_DIA_CONTENTS
BEGIN_REPORT_DGN_LIST
REPORTED-ON_DEST_DGN
BEGIN_REPORT_DEN_LIST
REPORTED-ON_DEST_DEN
SPEC_DIA_TYPE
SPEC_DIA_CONTENTS
END_REPORT_DEN_LIST
END_REPORT_DGN_LIST
   (other LLIDs may occur in FS2 flow, and will be truncated)
______________________________
Format Set 2 DTMU
  Server Object

Px...Cmd...DL... | Server Object |   Sx

FS2 DTMU with DIA-Report Coexistence Indicated by Agent=DIASTATUS
```

Figure 61. Coexistence Mapping of FS1 DIA Report to DTMU Server Object

Null RGN Handling

One subtle, but significant, difference between FS1 and FS2 involves RGNs. FS1 allows a "null RGN" to be used, in which no RGN value is given. A null RGN is encoded with an LT that has a length of 2 and no atomic data (see Appendix G for further details). The FS1 assumption is that, just as a RGN may have a value of, say, A, NET5, or USCO1N5, it may also have an empty (or null) string as its value. In this last case, the RGN is said to be *null.* FS2 does not allow null RGNs, but requires that a non-null RGN value be used in all MUs whenever a DSU name is specified. (Usually this RGN value will merely be the appropriate network ID.)

To allow FS1 implementations using null RGNs to coexist with FS2 implementations, which consider null RGNs to be illegal, the system administrator should define a specific (i.e., non-null) value to the FS2 implementations for the FS1 null RGN. This special RGN value is inserted by a bilingual DSU's FS1 DS_Receive process whenever a null RGN is received, and transformed into a null RGN by the bilingual DSU's FS1 DS_Send process.

This is most easily understood by the following example: Let "<null>" denote a null RGN, and suppose the system administrator has chosen NRGNVAL to be the FS2 name for the FS1 null RGN. User (*DGN*=DEPTA, *DEN*=ALICE) is located at an FS1-only DSU named (*RGN*= <null>, *REN*=SYSTEMA). User (*DGN*=DEPTB, *DEN*=BILL) is located at an FS2-only DSU named (*RGN*=NETWKB, *REN*=SYSTEMB). The bilingual DSU between the FS1 and FS2 subnets is known by two DSU names, (*RGN*=NETWKC, *REN*=SYSTEMC) and (*RGN*= <null>, *REN*=SYSTEMC) This network is shown in Figure 62.

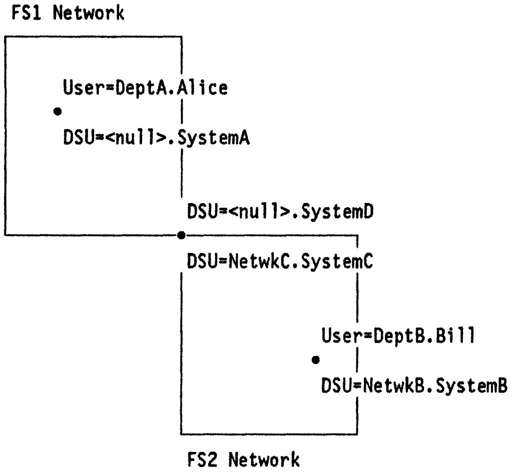


Figure 62. Null RGN Handling Example—Network, DSUs, and Users

The following system data structures exist in the various DSUs:

- At DSU (<null>.SYSTEMA)

  The directory contains

  ```
  DeptB.*        → <null>.SystemD
  DeptA.Alice    → local user queue #5
  ```

  The routing table contains

  ```
  <null>.SystemD → <null>.SystemD
  ```

- At the bilingual DSU (NETWKC.SYSTEMC)

  The directory contains

  ```
  DeptB.Bill     → NetwkB.SystemB
  ```

The routing table contains

```
NRGNVAL.SystemA → <null>.SystemA
NetwkB.SystemB  → NetwkB.SystemB
```

The intervention list contains

```
NRGNVAL.SystemD
```

The "null RGN value" is defined as NRGNVAL.

- At DSU (NETWKB.SYSTEMB)

  The directory contains

  ```
  DeptB.Bill      → local user queue #8
  DeptA.Alice     → NRGNVAL.SystemA
  ```

  The routing table contains

  ```
  NRGNVAL.*       → NetwkC.SystemC
  ```

These tables are shown in Figure 63 on page 400.

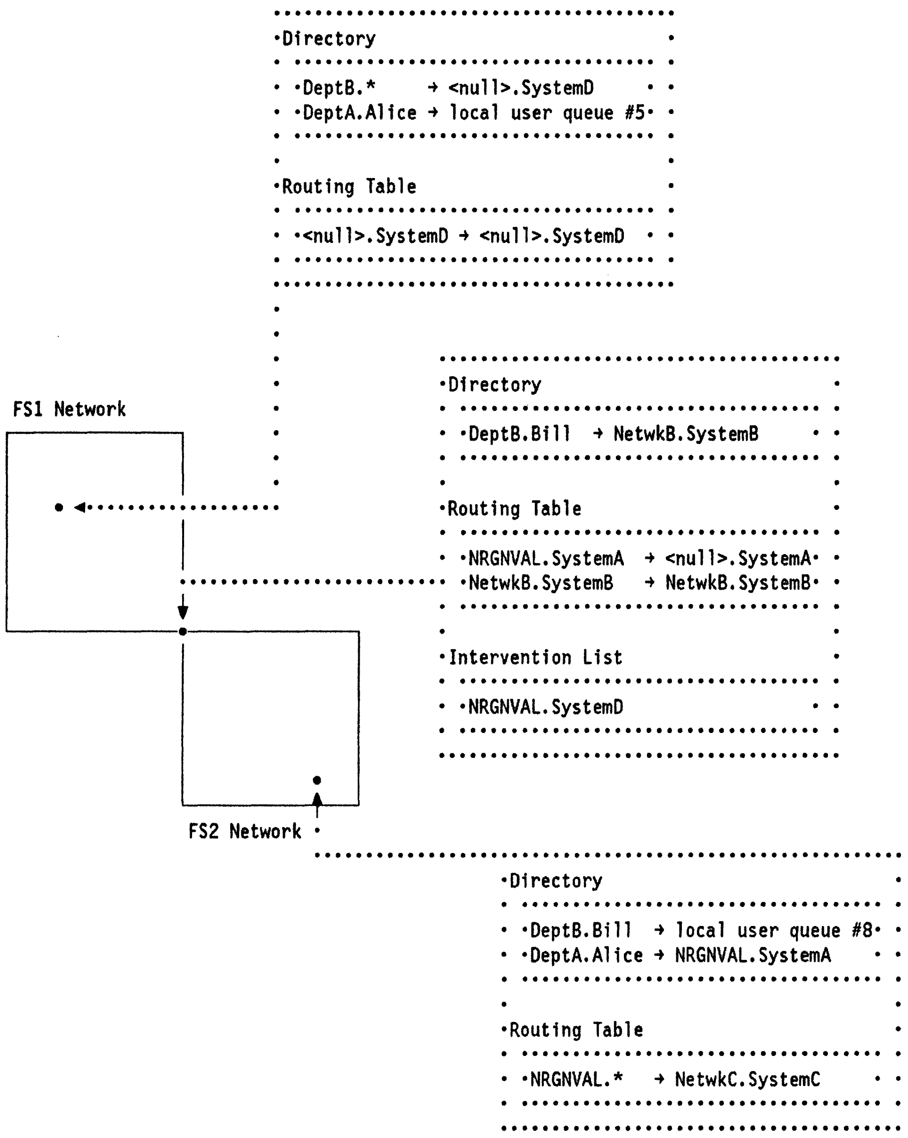

Directory
DeptB.* → <null>.SystemD
DeptA.Alice → local user queue #5
Routing Table
<null>.SystemD → <null>.SystemD
FS1 Network
Directory
DeptB.Bill → NetwkB.SystemB
Routing Table
NRGNVAL.SystemA → <null>.SystemA
NetwkB.SystemB → NetwkB.SystemB
Intervention List
NRGNVAL.SystemD
FS2 Network
Directory
DeptB.Bill → local user queue #8
DeptA.Alice → NRGNVAL.SystemA
Routing Table
NRGNVAL.* → NetwkC.SystemC

Figure 63. Null RGN Handling Example--Data Structures

If DEPTA.ALICE sends a distribution to DEPTB.BILL, the Dist_MU sent from the origin to the bilingual DSU will contain the following values.

```
Origin User: DGN=DeptA   DEN=Alice
Origin  DSU: RGN=<null>  REN=SystemA
Dest   User: DGN=DeptB   DEN=Bill
Dest    DSU: RGN=<null>  REN=SystemD
```

The distribution will be received by the bilingual DSU, and the null RGN values will be replaced with NRGNVAL in the FS1 DS_Receive process. The bilingual DSU will then redirect the distribution since the destination DSU (NRGNVAL.SYSTEMD) is in the DSU's intervention list. The directing sublayer will determine that the destination user, DEPTB.BILL, is located at NETWKB.SYSTEMB. The distribution as forwarded on to NETWKB.SYSTEMB from the bilingual DSU will have the following fields:

```
Origin User: DGN=DeptA   DEN=Alice
Origin  DSU: RGN=NRGNVAL REN=SystemA
Dest   User: DGN=DeptB   DEN=Bill
Dest    DSU: RGN=NetwkB  DEN=SystemB
```

The distribution will be received at NETWKB.SYSTEMB and delivered to DEPTB.BILL as would any other distribution. This example is shown in Figure 64.

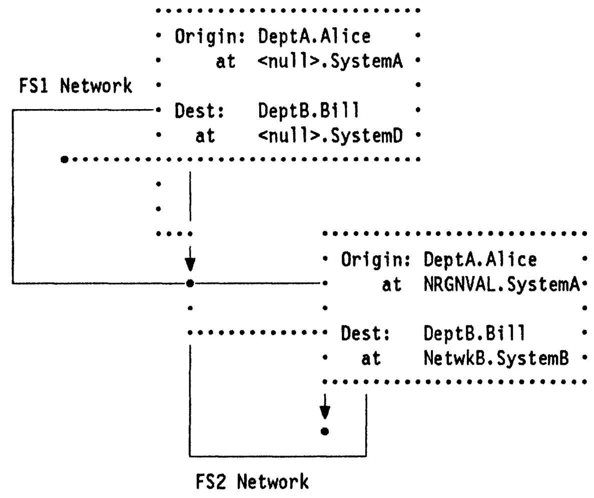


Figure 64. Null RGN Handling Example—Distribution Flow

Now suppose that DEPTB.BILL sends a distribution to DEPTA.ALICE. The distribution as originated from Bill's DSU will have the following fields:

```
Origin User: DGN=DeptB   DEN=Bill
Origin  DSU: RGN=NetwkB  REN=SystemB
Dest   User: DGN=DeptA   DEN=Alice
Dest    DSU: RGN=NRGNVAL REN=SystemA
```

At the bilingual DSU, the distribution will not be redirected. The *RGN*=NRGNVAL field will be converted to a null RGN by the bilingual DSU's FS1 DS_Send process, and the distribution as sent by the bilingual DSU and received by DEPTA.ALICE's DSU will have the following fields:

```
Origin User: DGN=DeptB   DEN=Bill
Origin  DSU: RGN=NetwkB  REN=SystemB
Dest   User: DGN=DeptA   DEN=Alice
Dest    DSU: RGN=<null>  REN=SystemA
```

Null RGNs in the *report_to_RGN* field are handled analogously.

FS1 Atomic Structures Not Present in FS2

The following Dist_MU *type* TRANSPORT atomic structures from FS1 are not mapped into FS2:

- *server_object_ind*

  This structure is unnecessary in FS2. The presence of a server object is signalled by the presence of the server object LLID.

- *server_parms*

  This structure was not used by FS1 applications.

The following FS1 Dist_MU *type* REPORT atomic structures are not mapped into FS2:

- *origin_seqno*

  This structure has a constant value in FS1 report flows. It may be assumed by bilingual DSUs.

- *agent_correl*

  This is *not* the same structure as *reported-on_agent_correl*. It was not used by FS1 applications, and is unnecessary in FS2 since reports are originated by the DSU (and no agent is involved to supply an *agent_correl*).

- *server_object_ind*

  This structure was not used by FS1 applications, and is unnecessary in FS2.

- *gen_SNADS_type*

  This structure is necessary in FS1 because DS and DIA reports may flow in a single MU. In FS2, only DS reports flow in the DRMU, and this structure is unnecessary.

- *spec_DS_** (all DS specific report fields)

  FS2 uses only general reports.

- *dist_flags*

  The appropriate values for these flags are assumed in FS2 without an explicit structure.

FS2 Atomic Structures Not Present in FS1

The following atomic structures in FS2 are not mapped into FS1.

- *MU_ID*

  This structure is not part of the distribution, but is used only for the hop-to-hop transfer of the DTMU or DRMU. FS1 uses the LU 6.2 Confirm/Confirmed verbs to transfer a DMU from one hop to the next.

- *MU_instance_number*

  This structure is used in FS2 only in conjunction with the *MU_ID*, discussed above.

The following DTMU atomic structures in FS2 are not mapped into FS1:

- *supplemental_dist_info1*
- *supplemental_dist_info2*

The following DRMU atomic structures in FS2 are not mapped into FS1:

- *reported-on_origin_RGN*

- *reported-on_origin_REN*
- *reported-on_dest_agent*
- *reported-on_supp_dist_info1*
- *reported-on_supp_dist_info2*
- *reported-on_dest_agent*
- *reported-on_origin_agent*
- *SNA_condition_report* (all atomic structures except *SNA_report_code*, *reported-on_dest_DGN* and *reported-on_dest_DEN*)

Appendix E. Exception Handling

This appendix describes protocols for reporting DS-detected conditions. In so doing, this appendix includes both references to DS encoding constructs described in Appendix G and references to the introduction on exception handling presented in Chapter 1.

Introduction

DS exception conditions can be detected at any of the four DS sublayers, and the exception can be reported in a variety of ways depending upon the detector and characteristics of the condition. This appendix identifies reportable conditions and describes the reporting and exception actions. For implementations where FS1 and FS2 functions coexist, exception-code translation tables are provided to translate the FS1 condition codes to the FS2 report codes. The remainder of this appendix presents a summary of exception conditions that are detected by the routing, directing, and transport sublayers. Exception conditions detected by the presentation services sublayer are described in detail in Appendix F and are not specifically addressed in this appendix.

Types of Reporting Actions

Given a reportable condition, the reporting action DS takes in response to the condition depends upon the detector and the characteristics of the condition. Listed below are the types of reporting actions DS may perform.

Local-Agent Reporting

Local reporting refers to reports that are delivered, either synchronously or asynchronously, only to the local agent (i.e., the agent local to the reporting DSU). Local reporting is used to report exception conditions that occur before DS has accepted responsibility for a distribution or when DS is performing an application-specific operation (i.e., a specific server operation at the origin or destination). The report is delivered to the agent across the agent protocol boundary.

MU-Level Reporting

DS informs the adjacent DSU whenever a condition is detected while in conversation with that DSU. MU-level reporting is performed when an exception is detected at the transport sublayer, and the condition is reported to the partner DSU by a SEMU (if DS_Send detects the exception) or by a REMU (if DS_Receive detects the exception). The specific protocols are defined in Chapter 2 and Chapter 3.

Distribution Reporting

Distribution reporting refers to the feedback on the condition of a distribution and is requested as part of the original distribution request. If the originator requests reporting on the condition of a distribution, DS will generate and send a distribution report to the report-to DSU/user whenever the responsible DSU has detected or has been informed of a condition that interferes with successful delivery of the distribution. The *SNA_condition_report* (SNACR) is provided as part of the distribution report and contains specific information about the exception. Since distribution reporting is requested as part of the distribution request, distribution reporting is appropriate only for DTMUs and DCMUs and is not performed for failed DRMUs or for any control MUs.

Note: If the report-to DSU/user is local to the reporting DSU, the distribution report is passed across the protocol boundary on the Receive_Distribution_Report verb. In the non-local case, the distribution report will flow through the network as a DRMU.

Local-Operator Reporting

Whenever an exception condition has triggered a distribution report or a local-operator report, the DSU logs the exception condition in the exception log and informs the local operator, if an operator is available at that DSU.

When the DSU is capable of rerouting distributions, the operator may be notified without creating a distribution report. In these cases, the operator is given an opportunity to reroute distributions that would fail in a less capable DSU. The following conditions fall into this category:

- Routing exception
- Function not supported
- System exception (condition is at adjacent node)
- MU sequence exception
- Repeated conversation problems

Note: The rerouting function gives the operator the ability to retry conditions that are not usually considered retriable (i.e., routing exception).

Exception Logging: When the DSU detects an exception condition, it creates an entry in the exception log that describes the condition encountered. The following information is logged for every condition.

- Name of the transaction program that detected the exception
- Date
- Time
- *SNA_report_code*
- Additional exception information

Depending upon the exception, the following additional information is logged whenever appropriate.

- *distribution_ID*
- MU_ID
- *reported-on_destinations*
- Session information
- LU 6.2 information
- SEMU
- REMU
- Implementation-specific information

Characteristics of Exception Conditions

Retriable Conditions

Reportable conditions are either retriable or non-retriable. Retriable conditions are not message-unit-specific. The same message unit might have been successfully processed had it arrived a moment earlier or later; therefore it is expected that the message unit can still be successfully processed when it is retried a moment later. Given their characteristic transience, it is not appropriate for these retriable conditions to immediately trigger a local report, distribution report, or operator notification.

If a retriable condition is detected by either DS_Send or DS_Receive, MU-level reporting is used to inform the partner about the condition encountered. Then the distribution may be retried again later. If DS_Receive detects the exception and determines that it is retriable, DS_Send may or may not choose to honor the receiver's recommended retry action. Only when the retry count has been exhausted does the responsible DSU generate a distribution report.

If the exception detected by DS_Receive is a temporary condition at the receiving DSU (i.e., temporary storage medium exception), DS_Send places a hold on the next-DSU queues for that connection. When the temporary condition has been resolved, an instance of DS_Receive at the receiving DSU is expected to allocate a conversation to DS_Send at the sending DSU. The attached instance of DS_Send releases the exception-hold and begins sending traffic again.

Non-Retriable Condition

Non-retriable conditions are message-unit-specific. The nature of the condition is such that the distribution cannot proceed without a change to the message unit. These conditions result in a distribution report, an operator notification, or a local report.

For a non-retriable condition detected by either DS_Send or DS_Receive, MU-level reporting is used to inform the partner of the exception and a distribution report is generated to inform the report-to DSU/user. For those conditions detected outside the transport sublayer, distribution reporting is performed (if requested). However, for conditions that are outside the responsibility of DS (i.e., specific server exceptions), distribution reporting is not appropriate (although the exception may be reported to the local agent).

Condition Scope

For each reportable condition, some portion of the distribution is affected. For each reportable condition and for all subcodes of the condition, the portion of the distribution affected and the TPs that may detect the exception are identified in Table 1.

Table 1. Portion of the Distribution Affected by a Reportable Condition and the Detecting TP

Condition	What's Affected					Detecting TP		
	DSU	Con- nection	Dist Copy	Local Dests	Dests	Send	Rcv	Rtr- Dir
Insufficient Resource	X		X			X	X	X
Specific Server Exception			X	X		X	X	X
Unknown Resource Name			X	X		X	X	X
System Exception	X	X	X			X	X	X
Unable to Register MU_ID		X	X			X	X	
Operator Intervention	X		X			X	X	X
Storage Medium Exception	X		X			X	X	X
Server Object Not Found			X	X		X	X	X
Function Not Supported			X		X		X	
DS Conversation Exception		X	X			X	X	
Unknown User Name				X	X		X	X
Format Exception		X	X			X	X	X
Unrecognized Message Unit			X			X	X	
Directing Exception				X	X		X	X
Improper DS Usage of LU 6.2	X	X	X			X	X	
MU Sequence Exception	X	X	X			X	X	
Invalid Restart Byte-Position			X				X	
Routing Exception			X		X		X	X
Hop Count Exhausted			X		X	X	X	X

SNACR Usage, DS Report Codes, and Reports

DS Usage of SNACR

The *SNA_condition_report* is a means by which exception information for any type of SNA condition can be reported. The SNACR contains an SNA-registered report code and a defined structure identification; therefore, the reason for the exception and the location at which the exception was detected can be fully described.

The following list identifies which of the highest-level structures within the *SNA_condition_report* are used by DS:

- *SNA_report_code*

  Is a 4-byte SNA-registered code that identifies the detected condition.

- *structure_report*

  Identifies the header and (conditionally) the contents of DS encoding constructs that are relevant to the specified *SNA_report_code.*

- *reported-on_dest_list*

Identifies the list of destinations affected by the *SNA_report_code*. This structure is required by DS in any *SNA_condition_report* that appears in a DRMU. This structure is precluded in the REMU.

- *supplemental_report*

  Contains the additional exception data that is relevant to the report code.

Further detail regarding DS's use of the *SNA_condition_report* is described in the sections that follow.

SNA-Registered DS Conditions

The following table provides a condensed description for each SNA-registered DS condition. The table identifies, for each DS condition, the corresponding SNA-registered report code, the contents of the SNACR report structures, and the recommended retry action. A formal description of the SNA-registered report codes and contents of the report structures can be found in the "Sense Data" chapter of the *SNA Formats* manual. Unless otherwise stated, the suggested retry actions for sender-detected conditions are identical to the receiver's recommended actions.

Table 2 (Page 1 of 5). SNA-Registered Report Codes for DS and SNACR Contents

Condition	**SNA Report Code**		**Structure Report**		**Supplemental Report**	**Suggested Retry Action**
	Code	**Subcode**	**Number of Reports**	**Structure Contents**		
Insufficient Resource						
Insufficient resource	0812	0011	0	N/A	N/A	Expected
Specific-Server Exception						
Specific-server exception	085A	0000	0	N/A	N/A	Precluded
Unknown Resource Name						
Server	085B	0001	0	N/A	Unknown Server Name	Precluded
Agent (See Note 4)	085B	0002	0	N/A	N/A	Precluded
System Exception						
Unidentifiable	085C	0000	0	N/A	N/A	Expected
Identifiable	085C	0001	0	N/A	N/A	Expected
Permanent	085C	0002	0	N/A	N/A	Precluded
Unable to Register MU_ID						
Duplicate MU_ID	085D	0001	0	N/A	MU_ID Registry Values	Precluded
MU_ID greater than expected	085D	0002	0	N/A	MU_ID Registry Values	Expected
MU_ID not accepted due to a temporary registry condition	085D	0003	0	N/A	N/A	Expected

Table 2 (Page 2 of 5). SNA-Registered Report Codes for DS and SNACR Contents

Condition	SNA Report Code		Structure Report		Supplemental Report	Suggested Retry Action
	Code	Subcode	Number of Reports	Structure Contents		
MU_ID not accepted due to a permanent registry condition	085D	0004	0	N/A	N/A	Precluded
MU_ID registry is not initialized	085D	0005	0	N/A	N/A	Precluded
Operator Intervention						
Suspending	085E	0001	0	N/A	N/A	Expected
Purging	085E	0002	0	N/A	N/A	Expected (See Note 6)
Storage-Medium Exception						
Temporary storage-medium exception	0879	0001	0	N/A	N/A	Expected
Permanent storage-medium exception	0879	0002	0	N/A	N/A	Precluded
Server Object Not Found						
Server object not found	08A6	0001	0	N/A	N/A	Precluded
Function Not Supported						
Service parameter not supported	1003	0016	0	N/A	Service triplet(s)	Precluded
Service parameter level not supported	1003	0017	0	N/A	Service triplet(s)	Precluded
Destination-role function not supported (See Note 3)	1003	0018	1	Data value	(See Note 5)	Precluded
All-role function not supported (See Note 3)	1003	0019	1	Data value	(See Note 5)	Precluded
Reserved	1003	001A	N/A	N/A	N/A	N/A
Unable to initiate agent (See Note 4)	1003	001B	0	N/A	N/A	Precluded
Function conflicts with FS1 encodings	1003	001C	1	Data value	N/A	Precluded
Reserved	1003	001D	N/A	N/A	N/A	N/A
Reserved	1003	001E	N/A	N/A	N/A	N/A
Multiple-destination traffic not supported	1003	001F	1	N/A	N/A	Precluded
DS Conversation Exception						
Unable to allocate a conversation	1008	6046	0	N/A	N/A	Expected
Detected a RESOURCE FAILURE	1008	6047	0	N/A	N/A	Expected
Detected a Deallocate Type(ABEND) return code	1008	6048	0	N/A	N/A	Expected

Table 2 (Page 3 of 5). SNA-Registered Report Codes for DS and SNACR Contents						
Condition	SNA Report Code		Structure Report		Supplemental Report	Suggested Retry Action
	Code	Subcode	Number of Reports	Structure Contents		
Unknown User Name						
Unknown user name	100A	0001	0	N/A	N/A	Precluded
Format Exception						
Required structure absent	100B	0001	1	N/A	N/A	Precluded
Precluded structure present	100B	0002	1	N/A	N/A	Precluded
Multiple occurrences of a non-repeatable structure	100B	0003	1	Data value of 2nd occurrence	N/A	Precluded
Excessive occurrences of a repeatable structure	100B	0004	1	Data value of max+1 occurrence	Max value	Precluded
Unrecognized structure present where precluded	100B	0005	1 (See Note 1)	Data value	N/A	Precluded
Length outside specified range	100B	0006	1	Data value	Max in range	Precluded
Length exception	100B	0007	1	Data value	N/A	Precluded
Required combination of structures absent	100B	0008	>1	N/A	N/A	Precluded
Precluded combination of structures present	100B	0009	>1	N/A	N/A	Precluded
Required combination of structures and data values absent	100B	000A	>1	Data value	N/A	Precluded
Precluded combination of structures and data values present	100B	000B	>1	Data value	N/A	Precluded
Unknown or unsupported data value	100B	000C	1	Data value	N/A	Precluded
Incompatible data values	100B	000D	>1	Data value	N/A	Precluded
Precluded character present	100B	000E	1 (See Note 2)	Data value	N/A	Precluded
Data value out of range	100B	000F	1	Data value	Max in range	Precluded
Segmentation occurred when precluded	100B	0010	1	N/A	N/A	Precluded
Precluded data value	100B	0011	1	Data value	N/A	Precluded
Recognized but unsupported structure	100B	0012	1	N/A	N/A	Precluded
None of several possible structures found	100B	0013	1 (See Note 1)	Data value possible	N/A	Precluded
Incorrect order of child structures found	100B	0014	1	Header of the parent structure	N/A	Precluded

Table 2 (Page 4 of 5). SNA-Registered Report Codes for DS and SNACR Contents

Condition	SNA Report Code		Structure Report		Supplemental Report	Suggested Retry Action
	Code	Subcode	Number of Reports	Structure Contents		
Unrecognized Message Unit						
Unrecognized message unit	100C	0001	1 (See Note 1)	Message unit	N/A	Precluded
Directing Exception (See Note 4)						
Agent name known but not supported for user destination	100E	0001	0	N/A	N/A	Precluded
Agent name known but not supported for DSU destination	100E	0002	0	N/A	N/A	Precluded
Agent name known but is not available	100E	0003	0	N/A	N/A	Precluded
Improper DS Usage of LU 6.2						
Improper DS usage of LU 6.2	100F	0001	0	N/A	N/A	Precluded
MU Sequence Exception						
MU_ID already terminated	1018	0001	0	N/A	N/A	Expected
MU_ID state mismatch	1018	0002	0	N/A	N/A	Precluded
Reserved	1018	0003	N/A	N/A	N/A	Precluded
Terminate conversation ignored	1018	0004	0	N/A	N/A	Precluded
RRMU not followed by a CHANGE_DIRECTION Indicator	1018	0005	0	N/A	N/A	Precluded
Invalid Restart Byte Position (RBP)						
RBP > 1 plus *last_byte_received* value.	1019	0001	0	N/A	RBP Values	Precluded
Byte-count restart elective is not supported.	1019	0002	0	N/A	RBP Values	Precluded
Unable to restart at indicated RBP.	1019	0003	0	N/A	RBP Values	Precluded
Routing Exception						
Unknown RGN	8019	0001	0	N/A	N/A	Precluded
Unknown RGN, REN combination	8019	0002	0	N/A	N/A	Precluded
Reserved	8019	0003	N/A	N/A	N/A	Precluded
No connection for specified service level	8019	0004	1	Service triplets	N/A	Precluded
Hop Count Exhausted						
Hop count exhausted	801C	0001	0	N/A	N/A	Precluded
Conditions for Specific FS1 Exceptions						
User names lost	0849	0000	0	N/A	N/A	Precluded

Table 2 (Page 5 of 5). SNA-Registered Report Codes for DS and SNACR Contents

Condition	SNA Report Code		Structure Report		Supplemental Report	Suggested Retry Action
	Code	Subcode	Number of Reports	Structure Contents		
Resource not available	084F	0000	0	N/A	N/A	Precluded
Server object size incompatible with capacity level	100D	0001	0	N/A	N/A	Precluded
REMU exception	1012	0000	0	N/A	N/A	Precluded
Invalid server parameters	1013	0000	0	N/A	N/A	Precluded

Notes:

1. For these conditions, the *sibling_list* may be included within the SNACR.
2. For these conditions, the *structure_byte_offset* and *structure_segment_number* may be included within the SNACR.
3. For these conditions, the *structure_byte_offset* is present whenever the corresponding *structure_contents* does not represent the beginning of the structure. The *structure_segment_number* is present if the reported-on structure is not contained in the first segment.
4. For these conditions, a *supplemental_report* is not present, because the DRMU contains the appropriate report fields.
5. For these conditions, the *supplemental_report* is present whenever the *structure_report* does not identify the unsupported functions.
6. If the partially received distribution is purged by an operator at the receiving DSU, retry is expected. However, if the operator purges the distribution copy from the sender's queue, retry is precluded.

Generating a Distribution Report

Whenever a condition is detected for which a distribution report is generated, certain information specified in the distribution request is used to create the distribution report. Listed below is an explanation of how a distribution report is constructed.

- The *hop_count* is set for the report.
- The *service_parms* in the DRMU are set to the specified *report_service_parms* in the DTMU. If they are absent, the *service_parms* are derived by the reporting DSU from the values specified in the DTMU.
- The *report-to_agent* is set to the *report-to_agent* value in the DTMU. If the *report-to_agent* is absent in the DTMU, it defaults to the *origin_agent* value.
- The *reporting_DSU* is set to the name of the DSU generating the report.
- The *report_DTM* is set to the date and time the report was generated.
- The *report-to_DSU_user* is set to the specified *report-to_DSU* and *report-to_user* values in the DTMU. If these fields are absent, it is set to the *origin_DSU* and *origin_user* values in the DTMU.
- If the report is to be sent to a DSU other than the *origin_DSU* specified in the DTMU, the *reported-on_origin_DSU* field is generated and is set to the *origin_DSU* value of the DTMU.

- If the report is to be sent to a user other than the *origin_user* specified in the DTMU, the *reported-on_origin_user* field is generated and is set to the *origin_user* value of the DTMU.
- The *reported-on_seqno_DTM* is set to the *seqno_DTM* value in the DTMU.
- The *reported-on_supp_dist_info1* field is set to the *supplemental_dist_info1* value in the DTMU.
- The *reported-on_agent_correl* is set to the *agent_correl* value in the DTMU.
- The *reported-on_origin_agent* is set to the value of the *origin_agent* if the *report-to_agent* value in the DRMU is different from the *origin_agent* value in the DTMU.
- The *reported-on_dest_agent* is set to the value of the *dest_agent* if *dest_agent* is specified in the DTMU.
- The *receiving_DSU* is set to the name of the DSU that detected the exception if different from the *reporting_DSU* name.
- The *SNA_condition_report* is constructed according to the type of condition detected.
- The *reported-on_supp_dist_info2* is set to the value of *supplemental_dist_info2* if it is present in the DTMU.

Bilingual Node: Mapping SNA Report Codes and FS1 Condition Codes

The following tables describe the translation of exception codes between Format Set 1 and Format Set 2. In FS1, the DS condition codes were described by a 2 byte, DS-unique code. The DS condition codes were very general and did not provide a detailed method of reporting exception conditions. In FS2, the exception conditions are described by a 4-byte, SNA-unique code, which consists of a primary code and subcode. By using SNA-registered report codes, exception conditions can be categorized and a more precise description of the condition can be reported.

Table 3 (Page 1 of 2). FS2 SNA Report Code to FS1 Condition Code Mappings

Condition	SNA Report Code		FS1 Condition Code
	Code	**Subcode**	
Insufficient Resource	0812	0011	0010
User Names Lost	0849	0000	000D
Resource Not Available	084F	0000	000E
Specific-Server Exception	085A	0000	0006
Unknown Resource Name			
Server	085B	0001	0007
Agent	085B	0002	0009
System Exception			
All subcodes	085C	ssss	000F

Table 3 (Page 2 of 2). FS2 SNA Report Code to FS1 Condition Code Mappings

Condition	SNA Report Code		FS1 Condition Code
	Code	Subcode	
Unable to Register MU_ID			
All subcodes	085D	ssss	000F
Operator Intervention			
All subcodes	085E	ssss	000C
Storage-Medium Exception			
All subcodes	0879	ssss	0011
Server Object Not Found			
All subcodes	08A6	ssss	000E
Function Not Supported			
All subcodes	1003	ssss	0005
DS Conversation Exception			
Allocation exception	1008	6046	000E
Resource Failure	1008	6047	000F
Deallocate Type(ABEND)	1008	6048	000F
Unknown User Name	100A	0001	0002
Format Exception			
All subcodes	100B	ssss	0004
Unrecognized Message Unit	100C	0001	0004
Server-Object Size Incompatible with Capacity Level	100D	0001	0013
Directing Exception			
All subcodes	100E	ssss	0002
Improper DS Usage of LU 6.2 PS	100F	0001	000F
REMU Exception	1012	0000	0012
Invalid Server Parameters	1013	0000	0008
MU Sequence Exception			
All Subcodes	1018	ssss	000F
Invalid Restart Byte-Position			
All Subcodes	1019	ssss	0004
Routing Exception			
All subcodes	8019	ssss	0001
Hop Count Exhausted	801C	0001	0003

Table 4. FS1 Condition Code to FS2 SNA Report Code Mappings

Condition	FS1 Condition Code	SNA Report Code	
		Code	Subcode
Routing Exception	0001	8019	0000
Unknown User Name	0002	100A	0001
Hop Count Exhausted	0003	801C	0001
Format Exception	0004	100B	0000
Function Not Supported	0005	1003	0000
Specific-Server Exception	0006	085A	0000
Unknown Resource Name			
Server	0007	085B	0001
Agent	0009	085B	0002
Invalid Server Parameters	0008	1013	0000
Operator Intervention			
Purging	000C	085E	0002
User Names Lost	000D	0849	0000
Resource Not Available	000E	084F	0000
System Exception	000F	085C	0000
Insufficient Resource	0010	0812	0011
Storage-Medium Exception	0011	0879	0001
REMU Exception	0012	1012	0000
Server-Object Size Incompatible with Capacity Level	0013	100D	0001

Note: If the condition was receiver-detected and the receiver provided *exception_and_reply_data* in the FS1 REMU, the *exception_and_reply_data* is encoded within the *supplemental_report* of the *SNA_condition_report* for FS2 reporting. If the condition was sender-detected or detected outside of a conversation context and the detecting component routinely generates *exception_and_reply_data* to log, the logged *exception_and_reply_data* is also encoded within the *supplemental_report* of the *SNA_condition_report* for FS2 reporting.

Exception Handling and Analysis

The tables that follow provide a summary of exception conditions that can be detected by DS_Send, DS_Receive, and DS_Router_Director. The conditions are presented in a logical processing order in which the exceptions could be detected. The analysis tables describe exception handling only for those distributions for which high integrity was specified. No examples of basic-integrity traffic are shown. In the analysis tables, each exception condition is associated with each of the following:

- The portion of the distribution affected by the condition. The exception may affect the entire distribution copy, all destinations, or local destinations; however, the condition may affect not only the distribution but also the connection between DSUs or an entire DSU.
- The appropriate retry and reporting responsibilities. This describes whether the condition is retriable or non-retriable and indicates the corresponding reporting actions.

- The registered *SNA_report_code* that identifies the condition.
- The exception handling actions to perform in order to recover from the exception.

Note: For a more detailed discussion about exception handling performed by DS_Send and DS_Receive refer to "Exceptions Detected by the Distribution Transport Sublayer" on page 89. For a discussion of server-related exceptions see "Servers and Objects" on page 40.

Exception Conditions Detected During the Sending Process

Table 5 (Page 1 of 5). Exception Processing When Conditions Are Detected by DS_Send

Exception Conditions	What's Affected	Retry / Reporting Actions	SNA Report Code	Other Exception Actions
Exception conditions detected by LU 6.2 when DS_Send attempts to allocate a conversation with DS_Receive.				
LU 6.2 was unable to allocate a conversation with DS_Receive because no sessions were available.	Connection	Retry is allowed. Log the exception.	1008-6046	Attempt to re-allocate the conversation.
LU 6.2 was unable to allocate a conversation with DS_Receive, because the parameters were incorrect, the TP was unknown, or a remote or local LU exception has occurred.	Connection	Retry is precluded. Log the exception and inform the operator.	1008-6046	Await operator action.
Exception conditions detected by LU 6.2 while DS_Send is in conversation with DS_Receive.				
DS_Send detects from LU 6.2 a return code of RESOURCE_FAILURE.	Connection	Retry is allowed. Log the exception, inform the operator, and send a SEMU to the partner.	1008-6047	Attempt to re-allocate the conversation.
		When retry is exhausted, log the exception, inform the operator, and send a SEMU to the partner.	1008-6047	Hold the queues and await operator action.
DS_Send detects from LU 6.2 a return code of Deallocate Type(ABEND).	Connection	Retry is allowed. Log the exception, inform the operator, and send a SEMU to the partner.	1008-6048	Attempt to re-allocate the conversation.
		When retry is exhausted, log the exception, inform the operator, and send a SEMU to the partner.	1008-6048	Hold the queues and await operator action.
DS_Send has detected that the receiving DSU has issued an improper sequence of LU 6.2 verbs.	Connection	Retry is precluded. Log the exception and inform the operator.	100F-0001	Issue Deallocate Type(ABEND).
Exception conditions detected by the DSU's queue service when reading an entry from the next-DSU queue.				
The storage medium detected a temporary failure.	DSU	Retry is allowed. Log the exception and inform the operator.	0879-0001	Attempt to perform the I/O operation again.

Table 5 (Page 2 of 5). Exception Processing When Conditions Are Detected by DS_Send

Exception Conditions	What's Affected	Retry / Reporting Actions	SNA Report Code	Other Exception Actions
The storage medium detected a permanent failure.	DSU	Retry is precluded. Log the exception and inform the operator.	0879-0002	Hold the next-DSU queues and await operator action.
Operator has suspended further processing for this connection.	DSU	Retry is allowed when the operator releases the connection. Log the exception.	085E-0001	Hold the next-DSU queue and await operator action.
Queue service encounters an exception that prevents further processing of this queue.	DSU	Retry is allowed. Log the exception and inform the operator.	085C-0001	Attempt to read the queue entry again.
		When retry is exhausted, log the exception, and inform the operator.	085C-0001	Hold the next-DSU queues and await operator action.
Exception conditions detected by the DSU when DS_Send attempts to process control MUs.				
An exception was encountered when attempting to enqueue the control MU on the control MU queue.	Connection	Retry is allowed. Log the exception.	085C-0001	Issue Deallocate Type(ABEND).
An exception condition was encountered when attempting to send a control MU.	Connection	Retry is allowed. Log the exception and inform the operator.	085C-0001	Hold the queues and issue Deallocate Type(ABEND).
An exception occurs that prevents further processing of this control MU.	Connection	Retry is allowed. Log the exception.	085C-0001	Issue Deallocate Type(ABEND).
A CRMU was received with an MU_ID IN-TRANSIT, SUSPENDED, COMPLETED, or PURGED state, but the MU_ID registry indicates a NOT-ASSIGNED state.	Connection	Retry is precluded. Log the exception and inform the operator.	1018-0002	Hold the queues and await operator action
A REMU was received for which the MU_ID is in NOT-ASSIGNED state.	Connection	Retry is precluded. Log the exception and inform the operator.	1018-0002	Hold the queues and await operator action
Exception conditions detected by the DSU when attempting to begin the MU transfer to the partner.				
An MU_ID registry exception occurred when attempting to assign an MU_ID to the DMU.	Connection	Retry is precluded. Log the exception and inform the operator.	085D-0004	Hold the next_DSU queues and issue Deallocate Type(ABEND).
Hop count is 0.	Dist Copy	Retry is precluded. Log the exception, inform the operator, send a SEMU to the partner, and deliver a dist report to the specified DSU/user.	801C-0001	Terminate the distribution copy.

Table 5 (Page 3 of 5). Exception Processing When Conditions Are Detected by DS_Send

Exception Conditions	What's Affected	Retry / Reporting Actions	SNA Report Code	Other Exception Actions
Local resources temporarily unavailable to continue transfer to the partner.	Dist Copy	Retry is allowed. Log the exception, inform the operator, and send a SEMU to the partner.	0812-0011	Attempt mid-MU restart when appropriate; otherwise, retry with new MU_ID.
		When retry is exhausted, log the exception, inform the operator, send a SEMU to the partner, and deliver a dist report to the specified DSU/user.	0812-0011	Terminate the distribution copy and hold the next-DSU queues.
Operator has purged this distribution copy. (See Note 1)	Dist Copy	Retry is precluded. Log the exception, send a SEMU to the partner, and deliver a dist report to the specified DSU/user.	085E-0002	Terminate the distribution copy.
Exception conditions detected by the builder when attempting to encode the DMU.				
The DMU cannot be encoded due to format exceptions.	Dist Copy	Retry is precluded. Log the exception, inform the operator, send a SEMU to the partner, and deliver a dist report to the specified DSU/user.	100B-ssss	Terminate the distribution copy.
Builder exception occurs that prevents further processing of the DMU.	Dist Copy	Retry is allowed. Log the exception, inform the operator, and send a SEMU to the partner.	085C-0001	Attempt mid-MU restart when appropriate; otherwise ,retry with new MU-ID.
		When retry is exhausted, log the exception, inform the operator, send a SEMU to the partner, and deliver a dist report to the specified DSU/user.	085C-0001	Terminate distribution copy and hold the next-DSU queues.
Exception conditions detected by the general server when attempting to read the server object.				
The storage medium detected a temporary failure. The server return code was I/O EXCEPTION.	DSU	Retry is allowed. Log the exception, inform the operator, and send a SEMU to the partner.	0879-0001	Attempt to perform the I/O operation again.
The storage medium detected a permanent failure. The server return code was I/O EXCEPTION.	DSU	Retry is precluded. Log the exception, inform the operator, send a SEMU to the partner, and deliver a dist report to the specified DSU/user.	0879-0002	Hold the next-DSU queues and await operator action.

Table 5 (Page 4 of 5). Exception Processing When Conditions Are Detected by DS_Send

Exception Conditions	What's Affected	Retry / Reporting Actions	SNA Report Code	Other Exception Actions
Insufficient local resources available to process the object. The server return code was TEMPORARY SERVER EXCEPTION.	Dist Copy	Retry is allowed. Log the exception, inform the operator, and send a SEMU to the partner.	0812-0011	Attempt mid-MU restart when appropriate; otherwise retry with new MU-ID.
		When retry is exhausted, log the exception, inform the operator, send a SEMU to the partner, and deliver a dist report to the specified DSU/user.	0812-0011	Terminate the distribution copy.
The server object specified in the distribution request cannot be found. The server return code was OBJECT NOT FOUND.	Dist Copy	Retry is precluded. Log the exception, inform the operator, send SEMU to the partner, and deliver a dist report to the specified DSU/user.	08A6-0001	Terminate the distribution copy.
Exception conditions detected only by the specific server when DS is attempting to read the server object. (Direct Fetch)				
The specific-server name specified in the distribution is unknown.	Dist Copy	Retry is precluded. Log the exception, inform the operator, and send a SEMU to the partner.	085B-0001	Terminate the distribution.
The specific server has detected an exception that prevents further processing of the distribution copy. The server return code was SPECIFIC-SERVER EXCEPTION.	Dist Copy	Retry is precluded. Log the exception, inform the operator, send a SEMU to the partner, and deliver a server report to the local agent.	085A-0000	Terminate the distribution.
Exception conditions detected by the DSU when decoding control MUs from DS_Receive.				
The Parser is unable to decode the control MU from DS_Receive.	Connection	Log the exception.	100B-ssss or 100C-0001	Issue Deallocate Type(ABEND).
The control MU received contained an instance number that is below the current instance number for the MU_ID.	Not Applicable	No retry or reporting actions.	No Code	Discard the control MU.
Exception conditions detected by the DSU's queue service when deleting an entry from the next-DSU queue following successful transmission of the DMU.				
The storage device detected a failure.	DSU	Log the exception and inform the operator.	0879-ssss	Await operator action.
Operator has suspended further processing for this connection.	DSU	Retry is suspended. Log the exception.	085E-0001	Hold the connection and await operator action.

Table 5 (Page 5 of 5). Exception Processing When Conditions Are Detected by DS_Send

Exception Conditions	What's Affected	Retry / Reporting Actions	SNA Report Code	Other Exception Actions
The queue service encounters an exception that prevents further processing of the distribution copy.	DSU	Log the exception and inform the operator.	085C-0001	Await operator action.

Notes:

1. Because the operator has purged the distribution copy from the sender's queue, retry is precluded.

Exception Conditions Detected During the Receiving Process

Table 6 (Page 1 of 6). Exception Processing When Conditions Are Detected by DS_RECEIVE.

Exception Conditions	What's Affected	Receiver's Retry Recommendations	SNA Report Code	Other Exception Actions
Exception conditions detected by LU 6.2 when DS_Receive attempts to allocate a conversation with DS_Send.				
LU 6.2 was unable to allocate a conversation with DS_Send because no sessions were available.	Connection	Retry is allowed. Log the exception and inform the operator.	1008-6046	Attempt to re-allocate the conversation.
LU 6.2 was unable to allocate a conversation with DS_Send, because the parameters were incorrect, the TP was unknown, or a remote or local LU exception has occurred.	Connection	Retry is precluded. Log the exception and inform the operator.	1008-6046	Stop DS_Receive and await operator action.
Exception conditions detected by LU 6.2 while DS_Receive is in conversation with DS_Send.				
DS_Receive detects from LU 6.2 a return code of RESOURCE_FAILURE.	Connection	Retry is allowed. Log the exception, inform the operator, and send a REMU to the partner.	1008-6047	Attempt to re-allocate the conversation.
		When retry is exhausted, log the exception, inform the operator, and send a REMU to the partner.	1008-6047	Await retry.
DS_Receive detects from LU 6.2 a return code of Deallocate Type(ABEND).	Connection	Retry is allowed. Log the exception, inform the operator, and send a REMU to the partner.	1008-6048	Attempt to re-allocate the conversation.
		When retry is exhausted, log the exception, inform the operator, and send a REMU to the partner.	1008-6048	Await retry.
DS_Receive has detected that the sending DSU has issued an improper sequence of LU 6.2 verbs.	Connection	Retry is precluded. Log the exception and inform the operator.	100F-0001	Issue Deallocate Type(ABEND).

Table 6 (Page 2 of 6). Exception Processing When Conditions Are Detected by DS_RECEIVE.

Exception Conditions	What's Affected	Receiver's Retry Recommendations	SNA Report Code	Other Exception Actions
Exception conditions detected by DS_RECEIVE while receiving DMUs.				
A DMU was received but the MU_ID has already been terminated.	Dist Copy	Retry is allowed. Log the exception, inform the operator, and issue the Send_Error verb. Optionally a REMU may be sent to the partner.	1018-0001	Discard the MU and continue as normal.
A DTMU was received with an MU_ID in the SUSPENDED state.	Dist Copy	Retry is precluded. Log the exception, inform the operator, and issue the Send_Error verb. Optionally a REMU may be sent to the partner.	1018-0002	Discard the MU, send all waiting control MUs, and deallocate the conversation.
A DTMU was received with an MU_ID in the COMPLETED or PURGED state.	Dist Copy	Retry is precluded. Log the exception and inform the operator. Optionally a Send_Error verb may be issued.	1018-0002	Discard the MU.
A DCMU was received with an MU_ID in the NOT-RECEIVED state.	Dist Copy	Retry is precluded. Log the exception, inform the operator, and issue the Send_Error verb. Optionally a REMU may be sent to the partner.	1018-0002	Discard the MU, send all waiting control MUs, and deallocate the conversation.
A DCMU was received with an MU_ID in the COMPLETED or PURGED state.	Dist Copy	Retry is precluded. Log the exception and inform the operator. Optionally a Send_Error verb may be issued.	1018-0002	Discard the MU.
DS_Receive has detected that the sending DSU has ignored a previous terminate-conversation indication.	Dist Copy	Retry is precluded. Log the exception, inform the operator, and issue the Send_Error verb. Optionally a REMU may be sent to the partner.	1018-0004	Discard the MU, send all waiting control MUs, and deallocate the conversation.
Exception Conditions detected by the parser when attempting to decode the DMU.				
The MU type is not recognized.	Dist Copy	Retry is precluded. Log the exception, inform the operator, issue the Send_Error verb, and send a REMU to the partner.	100C-0001	Discard the MU, send all waiting control MUs and deallocate the conversation.
The MU cannot be decoded due to format exceptions.	Dist Copy	Retry is precluded. Log the exception, inform the operator, and send a REMU to the partner.	100B-ssss	Discard the MU.

Table 6 (Page 3 of 6). Exception Processing When Conditions Are Detected by DS_RECEIVE.

Exception Conditions	What's Affected	Receiver's Retry Recommendations	SNA Report Code	Other Exception Actions
DMU specifies multiple destinations, but multiple destination traffic is not supported.	Dist Copy	Retry is precluded. Log the exception, inform the operator, and send a REMU to the partner.	1003-001F	Discard the MU.
Parser exception occurs that temporarily prevents further processing of the DMU.	Dist Copy	Retry is allowed. Log the exception, inform the operator, and send a REMU to the partner.	085C-0001	Await retry action from the sender.
Exception Conditions detected by the parser when attempting to decode a DCMU.				
The *restart_byte_position* value is 0.	Dist Copy	Retry is precluded. Log the exception, inform the operator, and send a REMU to the partner.	100B-000F	Discard the MU.
The *restart_byte_position* value is greater than one plus the *last_byte_received* value in the CRMU.	Dist Copy	Retry is precluded. Log the exception, inform the operator, and send a REMU to the partner.	1019-0001	Discard the MU.
The receiver does not support the byte-count restart elective and the RBP is not the beginning of the LLID following the last successfully received LLID.	Dist Copy	Retry is precluded. Log the exception, inform the operator, and send a REMU to the partner.	1019-0002	Discard the MU.
The byte-count restart elective is supported; the RBP ≠ 1; and the RBP ≤ *last_byte_received value* value in the CRMU, but the receiver is unable to restart at the indicated RBP. (See Note 3)	Dist Copy	Retry is precluded. Log the exception, inform the operator, and send a REMU to the partner.	1019-0003	Discard the MU.
Exception condition detected by the DSU's queue service when attempting to fetch a partially received DMU from the mid-MU restart queue.				
Unable to locate the partially received DMU in the mid-MU restart queue.	Dist Copy	Retry is precluded. Log the exception, inform the operator, and send a REMU to the partner.	085C-0002	Await retransmission with a new MU_ID.
Exception conditions detected by the DSU, when decoding the control information of the DMU, but before storing the object.				
A duplicate MU_ID has been received.	Dist Copy	Retry is precluded. Log the exception, inform the operator, and send a REMU to the partner.	085D-0001	Discard the MU.
The MU_ID is above the acceptable value.	Dist Copy	Retry is precluded. Log the exception, inform the operator, and send a REMU to the partner.	085D-0002	Discard the MU.

Table 6 (Page 4 of 6). Exception Processing When Conditions Are Detected by DS_RECEIVE.

Exception Conditions	What's Affected	Receiver's Retry Recommendations	SNA Report Code	Other Exception Actions
The DCMU received contained an instance number that was less than or equal to the current instance number.	Not Applicable	Issue the Send_Error verb. No retry or reporting actions.	No Code	Discard the DCMU.
The function requested is not supported by this DSU.	Dist Copy	Retry is precluded. Log the exception, inform the operator, and send a REMU to the partner.	1003-ssss	Discard the MU.
Local resources temporarily unavailable to receive the DMU from the partner.	Dist Copy	Retry is allowed. Log the exception, inform the operator, and send a REMU to the partner.	0812-0011	Await sender retry action.
The operator has purged the partially received distribution. (See Note 1)	Dist Copy	Retry is allowed. Log the exception and send a REMU to the partner.	085E-0002	Discard the partially received MU and await sender retry action.
Additional exception conditions detected by the DSU when receive-time enhancements are implemented.				
Hop count has reached 0 and one or more destinations are non-local.	Non-local Dests	Log the exception. Accept responsibility for the dist copy and deliver a dist report, to the specified DSU/user, for the terminated local dist copies.	801C-0001	Deliver the local copies and terminate the non-local copies.
This DSU does not provide the service level requested.	Dests	Retry is precluded. Log the exception, inform the operator, and send a REMU to the partner.	1003-0017	Discard the MU.
The agent or server is unknown at this DSU and some destinations are not local. (If all destinations are local, refer to Note 2.)	Local Dests	Log the exception. Accept responsibility for the dist copy and deliver a dist report to the specified DSU/user, for the terminated local dist copies.	085B-ssss	Forward the intermediate copies and terminate the local copies.
The agent is not supported for the destination DSU or user and some destinations are not local. (If all destinations are local, refer to Note 2.)	Local Dests	Log the exception. Accept responsibility for the dist copy and deliver a dist report to the specified DSU/user, for the terminated local dist copies.	100E-ssss	Forward the intermediate copies and terminate the local copies.
No entry was found in the routing table that satisfies the destination DSU name.	Some Dests	Log the exception. Accept responsibility for the dist copy and deliver a dist report, to the specified DSU/user, for the terminated dist copies.	8019-ssss	Forward copies containing valid dests and terminate copies containing invalid dests.

Table 6 (Page 5 of 6). Exception Processing When Conditions Are Detected by DS_RECEIVE.

Exception Conditions	**What's Affected**	**Receiver's Retry Recommendations**	**SNA Report Code**	**Other Exception Actions**
Destination user name is unknown to the local directory at the destination DSU, and the directory is exhaustive on the DGN.	Local Dests	Log the exception. Accept responsibility for the dist copy and deliver a dist report, to the specified DSU/user, for the terminated local copies.	100A-0001	Terminate the dist copies for the unknown local dests and forward the remaining valid dests.
Exception conditions detected by the general server when attempting to write the distribution object.				
The storage medium detected a temporary failure. The server return code was I/O EXCEPTION.	DSU	Retry is allowed. Log the exception, inform the operator, and send a REMU to the partner.	0879-0001	Discard the MU.
The storage medium detected a permanent failure. The server return code was I/O EXCEPTION.	DSU	Retry is precluded. Log the exception, inform the operator, and send a REMU to the partner.	0879-0002	Discard the MU.
Insufficient local resources were available to process the object. The server return code was TEMPORARY SERVER EXCEPTION.	Dist Copy	Retry is allowed. Log the exception, inform the operator, and send a REMU to the partner.	0812-0011	Await sender retry action.
The server attempted to write the server object, but no server object could be found. The server return code was OBJECT NOT FOUND	Dist Copy	Retry is precluded. Log the exception, inform the operator, and send a REMU to the partner.	08A6-0001	Discard the MU.
Exception conditions detected by the specific server when attempting to store the object. (Direct Store)				
The specific-server name specified in the distribution is unknown.	Local Dests	Retry is precluded. Log the exception, inform the operator, and send a REMU to the partner.	085B-0001	Discard the MU
The specific server has detected an exception that prevents further processing of the server object. The server return code was SPECIFIC-SERVER EXCEPTION.	Dist Copy	Mid-MU restart is expected at the LLID following the server object. Log the exception, inform the operator, and send a REMU to the partner. A server report will be delivered to the local agent.	085A-0000	Await DCMU from sender.
The specific server has failed to back out the server object.	Dests	Retry is precluded. Log the exception, inform the operator, and deliver a server report to the local agent.	085A-0000	Discard the MU.

Table 6 (Page 6 of 6). Exception Processing When Conditions Are Detected by DS_RECEIVE.

Exception Conditions	What's Affected	Receiver's Retry Recommendations	SNA Report Code	Other Exception Actions
Exception conditions detected by the DSU's queue service when enqueuing the DMU control block onto the router-director queue.				
The storage medium detected a temporary failure.	DSU	Retry is allowed. Log the exception, inform the operator, and send a REMU to the partner.	0879-0001	Discard the MU.
The storage medium detected a permanent failure.	DSU	Retry is precluded. Log the exception, inform the operator, and send a REMU to the partner.	0879-0002	Discard the MU.
Queue service encounters a temporary exception that prevents further processing of this queue.	Dests	Retry is allowed. Log the exception, inform the operator, and send a REMU to the partner.	085C-0001	Await sender retry action.
Exception condition detected by the DSU when scheduling the DS_Router_Director.				
DSU encounters an exception that prevents the scheduling of DS_Router_Director.	Dests	Retry is precluded. Log the exception, inform the operator, and send a REMU to the partner.	085C-0002	Discard the MU.
Exception conditions detected by the DSU when attempting to process control MUs.				
Parser was unable to decode the Control_MU from DS_Send.	Connection	Log the exception and inform the operator.	100B-ssss	Issue Deallocate Type(ABEND).
An exception was encountered when attempting to enqueue the control MU on the control MU queue.	Connection	Retry is allowed. Log the exception.	085C-0001	Issue Deallocate Type(ABEND).
An exception condition was encountered when attempting to send a control MU.	Connection	Retry is allowed. Log the exception and inform the operator.	085C-0001	Issue Deallocate Type(ABEND).
An exception occurs that prevents further processing of this control MU.	Connection	Retry is allowed. Log the exception and inform the operator.	085C-0001	Issue Deallocate Type(ABEND).
The control MU received contained an instance number that is below the current instance number for the MU_ID.	Not Applicable	No retry or reporting actions.	No Code	Discard the control MU.
An RRMU was followed by something other than a CHANGE_DIRECTION indicator (i.e., WHAT_RECEIVED=SEND).	Connection	Retry is precluded. Log the exception and inform the operator.	1018-0005	Issue Deallocate Type(ABEND).

Notes:

1. Because the operator has purged the distribution copy from the receiver's queue, retry is expected.
2. In cases where all destinations are local, the receiver may choose to log the exception, inform the operator, send a REMU to the partner, and then discard the partially received DMU.
3. In this exception case, the receiver may reject the DCMU or accept the DCMU by discarding the excessive bytes and continuing as normal.

Exception Conditions Detected while Performing Routing and Directing

Table 7 (Page 1 of 2). Exception Processing While Performing Routing and Directing

Exception Conditions	What's Affected	Reporting Responsibilities	SNA Report Code	Other Exception Actions
Exception conditions detected by the DSU's queue service when reading an entry from the router-director queue.				
The storage medium has detected a failure.	DSU	Log the exception, inform the operator, and deliver a dist report to the specified DSU/user.	0879-ssss	Terminate the distribution copy.
The queue service encounters an exception that prevents further processing of the distribution copy.	Dist Copy	Log the exception, inform the operator, and deliver a dist report to the specified DSU/user.	085C-0001	Terminate the distribution copy.
Exception conditions detected by the DSU when attempting to route the distribution to the next DSU.				
Hop count is 0 and one or more destinations are non-local.	Non-local Dests	Log the exception, inform the operator, and deliver a dist report to the specified DSU/user.	801C-0001	Deliver the local copies and terminate the non-local copies.
At this DSU, no connection is available for the specified service level.	Non-local Dests	Log the exception, inform the operator, and deliver a dist report to the specified DSU/user.	8019-0004	Deliver the local copies and terminate the non-local copies.
No entry was found in the routing table that satisfies the DSU name.	Some Dests	Log the exception, inform the operator, and deliver a dist report to the specified DSU/user.	8019-0001 or 8019-0002	Forward copies containing valid dests and terminate copies containing invalid dests.
Exception conditions detected by the DSU when attempting to perform local delivery at this DSU.				
Destination user name is unknown to the local directory at the destination DSU, and the directory is exhaustive on the DGN.	Local Dests	Log the exception, inform the operator, and deliver a dist report to the specified DSU/user.	100A-0001	Terminate the dist copies for the unknown local dests and forward the remaining valid dests.
The agent is unknown at this DSU.	Local Dests	Log the exception, inform the operator, and deliver a dist report to the specified DSU/user.	085B-0002	Terminate the dist copies for local dests and forward the dist copies for non-local dests.
The agent is known at the DSU, but is unavailable.	Local Dests	Log the exception, inform the operator, and deliver a dist report to the specified DSU/user.	100E-0003	Terminate the dist copies for local dests and forward dist copies for the non-local dests.

Table 7 (Page 2 of 2). Exception Processing While Performing Routing and Directing

Exception Conditions	What's Affected	Reporting Responsibilities	SNA Report Code	Other Exception Actions
The agent is known, but is not supported for the user or DSU destination.	Local Dests	Log the exception, inform the operator, and deliver a dist report to the specified DSU/user.	100E-0001 or 100E-0002	Terminate the dist copies for local dests and forward the dist copies for non-local dests.
Exception conditions detected by the general server when performing auxiliary operations.				
The storage medium has detected a failure.	DSU	Log the exception, inform the operator, and deliver a dist report to the specified DSU/user.	0879-ssss	Terminate the distribution copy.
Insufficient local resources available to process the object. The server return code was TEMPORARY SERVER EXCEPTION.	Dist Copy	Log the exception, inform the operator, and deliver a dist report to the specified DSU/user.	0812-0011	Terminate the distribution copy.
The server attempted to process the object, but no server object was found. The server return code was OBJECT NOT FOUND.	Dist Copy or Local Dests	Log the exception, inform the operator, and deliver a dist report to the specified DSU/user.	08A6-0001	Terminate the distribution copy.
Exception conditions detected by the specific server when performing auxiliary operations.				
The specific-server name specified in the distribution is unknown at this DSU.	Dist Copy	Log the exception and inform the operator. At the origin, deliver a local report to the originator. At the destination, deliver a dist report to the specified DSU/user.	085B-0001	At the origin, terminate the auxiliary operation and terminate the dist copy. At the destination, terminate the auxiliary operation and purge the distribution copy.
A specific-server exception occurs that prevents further processing of the distribution.	Dist Copy	Log the exception, inform the operator, and deliver a specific server report to the local agent.	085A-0000	At the origin, terminate the auxiliary operation and terminate the dist copy. At the destination, terminate the auxiliary operation and deliver the distribution.

Exception Handling for Format Set 1

This section provides an analysis of exception conditions that can be detected in the transport sublayer for FS1 protocols.

Exception Conditions

The tables that follow provide an exhaustive list of exception conditions that are detected by DS_Send and DS_Receive in the logical processing order in which they could be detected. (The abbreviations used in the tables are explained in the following discussion.) The mapping between exception conditions and their associated recovery actions is defined within the context of each exception occurrence as characterized by:

1. Which TP detected the exception?

   - DS_Send: The exception is detected during the sending process.
   - DS_Receive: The exception is detected during the receiving process.

2. Given that an exception is detected by DS_Send or DS_Receive, when is that exception detected with respect to the DMU transfer?

   - BT (Before Transfer): Either DS_Send has not yet issued the first Send_Data, or DS_Send is issuing LU 6.2 basic conversation verbs but the conversation has not yet been allocated (DS_Send is notified by an LU 6.2 PS return code).
   - DT (During Transfer): The conversation has been successfully allocated. DS_Send has issued its first Send_Data and DS_Receive has issued its first Receive_And_Wait.
   - AT (After Transfer): A complete DMU has been confirmed by the receiver, no new transfer has been initiated by the sender, and the conversation is still active.

3. What is the scope of the exception condition?

   - DMU: The exception condition affects the entire DMU and is unrelated to the list of destination users.
     - DMU(A): An LU 6.2 PS exception condition prior to conversation establishment will affect the first entry on the appropriate next-DSU queue if the retry count is exhausted.
   - ¬ALL Dest: One or more destination users or destination DSUs are directly affected by the exception condition, and one or more destination users or destination DSUs are unaffected.
   - ALL Dest: All destination users or all destination DSUs, for a particular DMU, are affected equally by the exception condition.
   - — (To be determined): A temporary exception condition will not affect the DMU (or DSU) unless, after an implementation-defined number of retries, the exception condition is considered to be unrecoverable.

     The actions required during retry are specified on the "—" line. If retry fails, the scope of, and actions for, the resulting permanent exception condition are specified on the following line.

> **Note:** Whenever all destination users or all destination DSUs are affected, the entire DMU is affected. A scope of ALL is distinguishable from a scope of the DMU only because ALL implies that the underlying exception condition is per destination but, in this case, all destinations are affected.

As a result of implementation choices concerning role, electives, and optimizations, the set of FS1 condition codes generated may vary slightly from one implementation to another. All implementations must be prepared to receive any of the valid FS1 condition codes (including any codes that the implementation does not generate).

Exception Actions

Exception conditions detected by the transport sublayers exception handling procedures result in the following five possible exception actions (the abbreviations defined below are used in the following tables):

- H/R: Hold/release queue
- Inform the adjacent DSU:
  - SEMU: Send a SEMU.
  - REMU: Send a REMU.
- Log/Msg: Log and notify operator.
- Report: Generate a distribution report.
- Purge: Remove the distribution request from the queue and purge the associated object file.

The tables in this appendix specify the appropriate exception actions for each FS1 exception condition occurring in a given context. Every exception action is classified by one of the following:

- N/A: The corresponding exception action is not applicable, given the exception condition and retry status, regardless of the exception scope.
- O: The corresponding exception action is neither required nor disallowed by FS1-defined exception-handling procedures. In these cases, the implementation decides whether or not the action will be performed.
- R: The corresponding exception action is required on behalf of all destination users specified in the DMU.
- R-1: The corresponding exception action is required on behalf of only those users directly affected by the exception condition.

If the set of appropriate actions for a particular exception condition includes sending of a distribution report, the exception table specifies the FS1 condition-code value that will be encoded into the resulting MU. An exhaustive list of the FS1 condition-code values and whether each code is valid in an REMU, Dist_MU type status, or both, is specified in Appendix G.

If the set of appropriate actions for a particular exception condition includes the sending of an REMU or SEMU, the exception table specifies the exception class that is encoded into the resulting transmission. The exception class for any sender-detected exception has the hexadecimal value C5. The exception class for a receiver-detected exception has one of the following values that will be used by the sender to determine sender actions:

- C2 or C3 (syntax or semantic exception, respectively): The nature of this class of exceptions is such that sender retry is not likely to succeed. Typically, the sender will dequeue the DMU, create a distribution report if requested, purge the DMU, log the error, and notify the operator.
- C4 (processing exception, unrelated to any particular DMU): Sender retry is recommended for this class of exceptions. Typically, the sender will hold the queues for an implementation-determined time interval, after which retransmission is attempted.

FS1 Exception Conditions Detected by DS_Send

Table 8 (Page 1 of 3). FS1 Exception Processing When Conditions are Detected by DS_Send

Exception Description	When	FS1 Cond Code	Excp Class	Scope	Exception Actions					
					H/R	SEMU	REMU	Log Msg	Report	Purge
Queue exception conditions when reading an entry from the next-DSU queue.										
I/O device experiences hardware failure.	BT	N/A	N/A	DSU	N/A	N/A	N/A	R	N/A	R
Any queue error occurs in the system that prevents further processing for this DMU (i.e., queue cannot be found).	BT	—	—	—	R	N/A	N/A	O	N/A	N/A
	BT	N/A	N/A	DSU	N/A	N/A	N/A	R	N/A	R
Builder exception conditions when encoding the command portion of the DMU.										
The DMU cannot be encoded.	BT	0004	N/A	DMU	N/A	N/A	N/A	R	R	R
	DT	0004	C5	DMU	N/A	R	N/A	R	R	R
LU 6.2 PS exception conditions when allocating a conversation with DS_Receive.										
Allocation parameters were incorrectly specified.	BT	000F	N/A	DMU(A)	N/A	N/A	N/A	R	R	R
LU 6.2 cannot allocate the requested conversation at this time because either there are no available sessions or LU 6.2 is temporarily unable to start an instance of DS_Receive.	BT	—	—	—	R	N/A	N/A	O	N/A	N/A
	BT	000E	N/A	DMU(A)	N/A	N/A	N/A	R	R	R
LU 6.2 is unable to allocate the requested conversation because of local or remote LU exceptions.	BT	—	—	—	R	N/A	N/A	O	N/A	N/A
	BT	000F	N/A	DMU(A)	N/A	N/A	N/A	R	R	R

Table 8 (Page 2 of 3). FS1 Exception Processing When Conditions are Detected by DS_Send

Exception Description	When	FS1 Cond Code	Excp Class	Scope	Exception Actions					
					H/R	SEMU	REMU	Log Msg	Report	Purge
LU 6.2 PS exception conditions when sending one or more DMUs across the conversation.										
Receives an LU 6.2 return code of RESOURCE_FAILURE or Deallocate Type(ABEND).	BT, DT, or AT	—	—	—	R	N/A	N/A	O	N/A	N/A
	BT, DT, or AT	000F	N/A	DMU	N/A	N/A	N/A	R	R	R
The receiving DSU has violated the FS1 protocols regarding usage of LU 6.2.	DT	—	—	—	R	N/A	N/A	O	N/A	N/A
	DT	000F	N/A	DMU	N/A	N/A	N/A	R	R	R
Parser exception conditions when decoding the REMU.										
The parser is unable to decode the REMU.	DT	–	–	–	R	N/A	N/A	0	N/A	N/A
	DT	0012	N/A	DMU	N/A	N/A	N/A	R	R	R
Specific-server exception conditions when attempting to read the server object.										
Origin server cannot be found at the origin DSU.	DT	0007	C5	DMU	N/A	R	N/A	R	R	R
Origin-server parameters were incorrectly specified.	DT	0008	C5	DMU	N/A	R	N/A	R	R	R
Object contents are incompatible with file server.	DT	0006	C5	DMU	N/A	R	N/A	R	R	R
General-server exception conditions when attempting to read the server object.										
General server cannot be found at this DSU.	DT	000F	C5	DMU	N/A	R	N/A	R	R	R
General-server parameters were incorrectly specified.	DT	000F	C5	DMU	N/A	R	N/A	R	R	R
Any server exception occurs that permanently prevents further processing for this DMU.	DT	000F	C5	DMU	N/A	R	N/A	R	R	R
Specific- or general-server exception conditions when reading the server object.										
Server experiences a temporary condition that is expected to reset shortly so that retransmitting of the DMU can be attempted (i.e., another process currently has the data set open or locked).	DT	—	C5	—	R	R	N/A	O	N/A	N/A
	DT	0010	C5	DMU	N/A	R	N/A	R	R	R
I/O device experiences hardware failure.	DT	0011	C5	DMU	N/A	R	N/A	R	R	R
Specific-server exception conditions when attempting to decrement the object lock count after transmission is complete.										
Origin server not found at the origin DSU.	AT	N/A	N/A	DMU	N/A	N/A	N/A	0	N/A	N/A

Table 8 (Page 3 of 3). FS1 Exception Processing When Conditions are Detected by DS_Send

Exception Description	When	FS1 Cond Code	Excp Class	Scope	Exception Actions					
					H/R	SEMU	REMU	Log Msg	Report	Purge
Origin-server parameters were incorrectly specified.	AT	N/A	N/A	DMU	N/A	N/A	N/A	0	N/A	N/A
Any server exception occurs that permanently prevents further processing for this DMU.	AT	N/A	N/A	DMU	N/A	N/A	N/A	0	N/A	N/A
General-server exception conditions when attempting to decrement the object lock count after transmission is complete.										
General server not found.	AT	N/A	N/A	DMU	N/A	N/A	N/A	0	N/A	N/A
General-server parameters were incorrectly specified.	AT	N/A	N/A	DMU	N/A	N/A	N/A	0	N/A	N/A
Any server exception occurs that permanently prevents further processing for this DMU.	AT	N/A	N/A	DMU	N/A	N/A	N/A	0	N/A	N/A
Specific- or general-server exception conditions when decrementing the object lock count after transmission is complete.										
I/O device experiences hardware failure.	AT	N/A	N/A	DMU	N/A	N/A	N/A	0	N/A	N/A
Any server error occurs in the system that temporarily prevents further processing for this DMU.	AT	N/A	N/A	DMU	N/A	N/A	N/A	0	N/A	N/A
Queue exception conditions when deleting an entry from the next-DSU queue following a successful transmission of a DMU.										
I/O device experiences hardware failure.	AT	N/A	N/A	DSU	N/A	N/A	N/A	0	N/A	N/A
Any queue error occurs in the system that prevents further processing for this DMU (i.e., queue cannot be found or in-use flag is not on).	AT	N/A	N/A	DSU	N/A	N/A	N/A	0	N/A	N/A

FS1 Exception Conditions Detected by DS_Receive

Table 9 (Page 1 of 3). FS1 Exception Processing When Conditions Are Detected by DS_Receive

Exception Description	When	FS1 Cond Code	Excp Class	Scope	Exception Actions					
					H/R	SEMU	REMU	Log Msg	Report	Purge
LU 6.2 PS exception conditions when allocating a conversation with DS_Send.										
Allocation parameters were incorrectly specified.	BT	N/A	N/A	DSU	N/A	N/A	N/A	0	N/A	N/A
LU 6.2 cannot allocate the requested conversation at this time because either there are no available sessions or LU 6.2 is temporarily unable to start an instance of DS_Receive.	BT	N/A	N/A	DSU	N/A	N/A	N/A	0	N/A	N/A
LU 6.2 is unable to allocate the requested conversation because of other local or remote LU exceptions.	BT	N/A	N/A	DSU	N/A	N/A	N/A	0	N/A	N/A
LU 6.2 PS exception conditions when receiving one or more DMUs across the conversation.										
Receives an LU 6.2 return code of RESOURCE_FAILURE or Deallocate Type(ABEND).	BT, DT, or AT	N/A	N/A	DSU	N/A	N/A	N/A	0	N/A	R
The sending DSU has violated the FS1 protocols regarding usage of LU 6.2.	DT or AT	N/A	N/A	DSU	N/A	N/A	N/A	R	N/A	R
DSU exception conditions when performing mandatory receive-time checks.										
Structure of the DMU does not match Appendix G.	DT	0004	C2	DMU	N/A	N/A	R	R	N/A	R
This DSU does not provide a service level specified as mandatory for a particular DSU.	DT	0005	C3	DMU	N/A	N/A	R	R	N/A	R
DSU exception conditions when receiver wishes to temporarily restrict the traffic it accepts based on an implementation-dependent cause. In these cases, the exception object will always be the command, encoded as X'07'. See Appendix G.										
A resource is not available.	DT	000E	C4	DMU	N/A	N/A	R	0	N/A	R
DSU exception conditions when receive-time enhancements are implemented.										
Hop count has reached 0 and one or more destination users are non-local.	DT	N/A	N/A	¬ALL Dest	N/A	N/A	N/A	N/A	N/A	N/A
	DT	0003	C3	All Dest	N/A	N/A	R	R	N/A	R
Destination transaction program is unknown at the destination DSU.	DT	N/A	N/A	¬ALL Dest	N/A	N/A	N/A	N/A	N/A	N/A
	DT	0009	C3	All Dest	N/A	N/A	R	R	N/A	R
Destination user name is unknown to the directory at the destination DSU. (See Note 1)	DT	N/A	N/A	¬All Dest	N/A	N/A	N/A	N/A	N/A	N/A
	DT	0002	C3	All Dest	N/A	N/A	R	R	N/A	R

Table 9 (Page 2 of 3). FS1 Exception Processing When Conditions Are Detected by DS_Receive

Exception Description	**When**	**FS1 Cond Code**	**Excp Class**	**Scope**	**Exception Actions**					
					H/R	**SEMU**	**REMU**	**Log Msg**	**Report**	**Purge**
Destination DSU name not found in routing table, and, therefore, next-DSU cannot be determined.	DT	N/A	N/A	¬ALL Dest	N/A	N/A	N/A	N/A	N/A	N/A
	DT	0001	C3	All Dest	N/A	N/A	R	R	N/A	R
DSU exception condition when the object size, as perceived by the receiving DSU, exceeds the requested capacity service level.										
Server object size is incompatible with capacity level.	DT	0013	C3	DMU	N/A	N/A	R	R	N/A	R
Specific-server exception conditions when receive-time enhancements are supported and direct storing of the object, using the destination server, is attempted.										
Server specified in the DMU is unknown at the destination.	DT	N/A	N/A	¬ALL Dest	N/A	N/A	N/A	N/A	N/A	N/A
	DT	0007	C3	All Dest	N/A	N/A	R	R	N/A	R
Destination-server parameters were incorrectly specified.	DT	N/A	N/A	¬ALL Dest	N/A	N/A	N/A	N/A	N/A	N/A
	DT	0008	C3	All Dest	N/A	N/A	R	R	N/A	R
Object contents are incompatible with file server.	DT	N/A	N/A	¬ALL Dest	N/A	N/A	N/A	N/A	N/A	N/A
	DT	0006	C3	All Dest	N/A	N/A	R	R	N/A	R
General-server exception conditions when attempting to write the distribution object.										
General server cannot be found at this DSU.	DT	000F	C4	DMU	N/A	N/A	R	0	N/A	R
General-server parameters were incorrectly specified.	DT	000F	C4	DMU	N/A	N/A	R	0	N/A	R
Any server exception occurs that permanently prevents further processing for this DMU.	DT	000F	C4	DMU	N/A	N/A	R	0	N/A	R
Specific- or general-server exception conditions when writing the distribution object.										
Temporarily unable to write due to insufficient capacity.	DT	000E	C4	DMU	N/A	N/A	R	0	N/A	R
Server experiences a temporary condition that is expected to reset shortly so that retransmitting of the DMU can be attempted (i.e., another process currently has the data set open or locked).	DT	0010	C4	DMU	N/A	N/A	R	0	N/A	R
I/O device experiences hardware failure.	DT	0011	C4	DMU	N/A	N/A	R	0	N/A	R
Queue exception conditions when enqueueing the DMU control block onto the router-director queue.										
I/O device experiences hardware failure.	DT	0011	C4	DMU	N/A	N/A	R	0	N/A	R

Table 9 (Page 3 of 3). FS1 Exception Processing When Conditions Are Detected by DS_Receive

Exception Description	When	FS1 Cond Code	Excp Class	Scope	Exception Actions					
					H/R	SEMU	REMU	Log Msg	Report	Purge
Any queue error occurs in the system that prevents further processing for this DMU (i.e., queue cannot be found).	DT	000F	C4	DMU	N/A	N/A	R	0	N/A	R
DSU exception conditions when scheduling DS_Router_Director.										
DS_Router_Director is temporarily unavailable.	DT	000E	C4	DMU	N/A	N/A	R	0	N/A	R
Any other DSU exception occurs that prevents the scheduling of DS_Router_Director.	DT	000F	C4	DMU	N/A	N/A	R	0	N/A	R
Parser exception conditions when decoding a SEMU.										
The parser is unable to decode the SEMU.	DT	N/A	N/A	DMU	N/A	N/A	N/A	0	N/A	N/A

Note:

1. An end-only-role DSU specialized in accepting single-destination traffic is permitted to send a REMU with the specified FS1 condition code if a multiple-destination DMU is received.

Exception Codes for a SEMU (Type FS1)

If DS_Send detects an exception while sending a DMU, DS_Send notifies DS_Receive by sending a SEMU (Type FS1). The SEMU (Type FS1) contains the exception information pertaining to the condition and indicates to the receiver that the remainder of the DMU will not be transmitted. The following table illustrates the exceptions that DS_Send may detect and the corresponding contents of the SEMU (Type FS1).

Error Condition	Exception Code			Sender Action
	Exception Class	Exception Condition Code	Exception Object	
System Exception	C5	06	(See page 623)	Recoverable
Insufficient Resource	C5	0B	1B	Recoverable
Storage-Medium Exception	C5	0B	(See page 623)	Unrecoverable
Specific-Server Exception	C5	18	1B	Unrecoverable
Format Exception	C5	0F	07	Unrecoverable

Note: Refer to Appendix G, page 623, for a description of the exception code GDS values and for the allowable values for the exception object.

Exception Codes for a REMU (Type FS1)

If DS_Receive detects an exception while receiving a DMU, DS_Receive notifies DS_Send by sending a REMU (Type FS1). The REMU (Type FS1) contains the exception information pertaining to the condition and the DSU name of the receiver. The following table illustrates the exceptions that DS_Receive may detect and the corresponding contents of the REMU (Type FS1).

Error Condition	FS1 Condition Code	Exception Code				Sender Action
		Exception Class	Exception Condition Code	Exception Object	Exception Data	
Routing Exception	0001	C3	02	09	Dest DSU and Service Parms	Unrecoverable
Unknown User Name	0002	C3	02	09	DGN and DEN	Unrecoverable
Hop Count Exhausted	0003	C3	11	09	—	Unrecoverable
Format Exception	0004	C2	(See page 625)	(See page 625)	LLIDF Received	Unrecoverable
Function Not Supported	0005	C3	(See page 625)	(See page 625)	LLIDF Received	Unrecoverable
Specific-Server Exception	0006	C3	18	1B	—	Unrecoverable
Unknown Resource Name - Server	0007	C3	02	1A	DOP	Unrecoverable
Invalid Server Parameters	0008	C3	17	1A	DOP	Unrecoverable
Unknown Resource Name - Agent	0009	C3	02	09	Agent TP	Unrecoverable
Resource Not Available	000E	C4	04	(See page 625)	-	Recoverable
System Exception	000F	C4	06, 0B	(See page 625)	-	Recoverable
Insufficient Resource	0010	C4	04, 0B	1B	-	Recoverable
Storage-Medium Exception	0011	C4	0B	(See page 625)	-	Recoverable
Server Object Size Incompatible with Capacity Level	0013	C3	11	1B	-	Unrecoverable

Note: Refer to Appendix G, page 625, for a description of the exception code GDS values and for the allowable values for the exception object.

Appendix F. Protocol Boundary Definitions

Introduction

The term *protocol boundary* is used generally to refer to the semantic definition of the data and control exchanges between two components in an SNA node. This appendix focuses on the protocol boundaries between DS and the application transaction programs that interact with DS. These are the agents that request services, the servers that work with DS to fetch and store objects, and agents that control the DS operation.

These protocol boundaries are described generically here in the form of precisely defined verbs. IBM products implementing DS do not necessarily provide a DS programming interface; however, the DS functions performed by the implementations are equivalent to the functions described in this appendix. For information about the correspondence between an implementation's DS functions and the protocol boundaries described in this appendix, refer to the appropriate IBM product publication. For an introductory discussion on the protocol boundary, refer to Chapter 1.

The DS *protocol boundary verbs* are formally described in verb description tables and parameter descriptions. A *verb description table* contains the primary syntax for each parameter associated with a particular verb. A *parameter description* contains a prose description of the parameter, enumeration values, and any conditions of presence associated with a particular parameter.

Verb Description Table

Column Descriptions

Supplied Parameter Name

The first column of the verb description tables, which begin on page 441, identifies the parameters supplied on the invocation of a particular verb, and illustrates their hierarchical relationship by indentation of the column entries. For example, in the Send_Distribution table on page 449, *Report-To_RGN* is shown indented underneath *Report-To_DSU*; this shows that *Report-To_RGN* is a child parameter contained by the *Report-To_DSU* parent parameter.

Returned Parameter Name

The *returned_parameter name* identifies the parameters returned as a result of the invocation of a particular verb, and illustrates their hierarchical relationship by indentation of the column entries.

Parameter Reference Page (Parm Ref Page)

As syntax is described in the particular verb description table, the semantics, enumeration values, and other characteristics are described formally in the parameter description. The *parameter reference page* column in the verb description table contains a reference page number to where this parameter information is found.

Length

The range of length values specifies the minimum and maximum lengths of parameters that an implementation is required to accept across the protocol boundary. Sometimes the length is described as an enumeration (ENUM), which means the parameter may be implemented as an integer, character string, pointer, or any implementation choice.

Occurrences

Multiple occurrences of parameters may or may not be permitted. A value of "1 - <some number>" in this column indicates the allowed range of occurrences of the corresponding parameter. A value of "≥1" indicates that there is no architecturally defined maximum. A value of "1" in this column indicates that only a single instance of the corresponding parameter is appropriate. A value of "0 - 1" indicates that an instance of the corresponding parameter is optional.

Note: An asterisk denotes special presence rules for a particular parameter. These presence rules are detailed in the corresponding parameter description.

Children

Number (Num): Each parent parameter contains a certain number of different children. This column specifies the minimum and maximum number of different children for a particular parent parameter. This column also specifies mutual exclusion among a set of optional children. If all children are optional ("0-1") and the parent parameter contains "1" for children number, one of the set of children occurs within that particular parent. This column does not account for multiple occurrences of a particular child within the parent parameter. Multiple occurrences of a particular parameter are indicated in the "Occurrences" column.

Subtable (Subtab): Sometimes the need to divide large tables into subtables becomes apparent, particularly when common parameters appear frequently within different parameter description tables. This column contains a reference page number to the page on which these common parameters are described.

Parameter Description

This description is referenced by a page number appearing in the "Parameter Reference Page" column corresponding to each parameter in the verb description table. The parameter description (see page 487, for example) contains information pertaining to a particular parameter. Prose descriptions, presence rules, enumeration values, and semantics associated with the corresponding entry in the verb description table may appear in the parameter description.

Distribution Verbs

VERB: Obtain_Local_Server_Report

Obtain_Local_Server_Report is issued by an agent to obtain a report generated by a specific server. It is used for those server reports that can not be returned on the normal sequences on Send_Distribution, Receive_Distribution, or Query_Distribution_Sending. Typically, these are reports of server problems, and are not related to a specific distribution.

Table 10. Obtain_Local_Server_Report

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Receiving_Agent	529	1-8	1	—	—
Distribution_Queue_ID	505	—	1	1	482
Queue_Entry_ID	528	32	0-1 [1]	—	—
Returned Parameter Name					
Return_Code [2]	540	ENUM	1	—	—
Queue_Entry_ID	528	32	0-1 [1]	—	—
Specific_Server_Report	549	1-32767	1	—	—

Notes:

1. When *queue_entry_ID* is used as a supplied parameter, it is not returned. If it is not used as a supplied parameter, then it is returned.
2. If the value of *return_code* is not OK, then all other expected parameters are not returned, except *specific_server_report* may still be returned.

VERB: Query_Distribution_Sending

Query_Distribution_Sending is issued by a sending agent to determine the state of the sending operations for the specified distribution.

This verb is required in several sending sequences and can also be issued in a stand-alone sequence when an agent wishes to determine if the distribution has been sent.

Table 11. Query_Distribution_Sending

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Distribution_ID	504	—	1	4-5	478
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—
Sending_State	544	ENUM	1	—	—
Specific_Server_Report	549	1-32767	0-1	—	—

VERB: Receive_Distribution

Receive_Distribution receives a distribution, either the first one in the specified local delivery queue, or a particular one of the distributions in the queue. A specific copy of a distribution being received is identified uniquely by three parameters. They are the *distribution_ID*, the *distribution_queue_ID*, and the *queue_entry_ID*. Supplying *distribution_queue_ID* retrieves the first entry from the specified local delivery queue. Supplying either *queue_entry_ID* or *distribution_ID* in addition to *distribution_queue_ID* retrieves a specific entry from the specified local delivery queue.

Table 12 (Page 1 of 2). Receive_Distribution

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Receiving_Agent	529	1-8	1	—	—
Distribution_Queue_ID	505	—	1	1	482
Distribution_ID	504	—	0-1[1]	4-5	478
Queue_Entry_ID	528	32	0-1[2]	—	—

Table 12 (Page 2 of 2). Receive_Distribution

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Returned Parameter Name					
Return_Code [3]	540	ENUM	1	—	—
Dest_Agent	500	1-8	0-1	—	—
Distribution_ID	504	—	0-1 [1]	4-5	478
Agent_Correl	489	1-128	0-1	—	—
Service_Parms	547	—	1	8	485
Destination	503	—	1-256	1-2	476
Agent_Object	490	1-32763	0-1	—	—
Server	545	1-8	0-1	—	—
Server_Access	545	1-64	0-1*	—	—
Server_Object_Byte_Count	547	8	0-1	—	—
Specific_Server_Info	549	1-32767	0-1*	—	—
Exception_Report_Req	506	ENUM	1	—	—
Report-To_DSU	532	—	0-1	2	—
Report-To_RGN	533	1-8	1	—	—
Report-To_REN	532	1-8	1	—	—
Report-To_User	533	—	0-1	2	—
Report-To_DGN	532	1-8	1	—	—
Report-To_DEN	531	1-8	1	—	—
Report-To_Agent	531	1-8	0-1	—	—
Report_Service_Parms	534	—	0-1	6	482
Specific_Server_Report	549	1-32767	0-1	—	—
Previously_Received_Date_Time	525	—	0-1	2	—
Previously_Received_Date	524	—	1	3	476
Previously_Received_Time	525	—	1	5	487
Queue_Entry_ID	528	32	0-1 [2]	—	—
Distribution_Time	505	—	1	5	487
Integrity	509	ENUM	1	—	—

Notes:

1. When *distribution_ID* is used as a supplied parameter, it is not returned. If it is not used as a supplied parameter, it is returned.
2. When *queue_entry_ID* is used as a supplied parameter, it is not returned. If it is not used as a supplied parameter, it is returned.
3. If the value of *return_code* is not OK, then all other expected parameters are not returned.

VERB: Receive_Distribution_Report

Receive_Distribution_Report is issued by a specially started instance of the report-to agent (which will often have defaulted to the origin agent) to receive a report on the DS condition of a distribution. A distribution report being received is identified uniquely by three parameters. They are the *distribution_ID*, the *distribution_queue_ID*, and the *queue_entry_ID*. Supplying *distribution_queue_ID* retrieves the first entry from the specified local delivery queue. Supplying either *queue_entry_ID* or *distribution_ID* in addition to *distribution_queue_ID* retrieves a specific entry from the specified local delivery queue.

Table 13 (Page 1 of 2). Receive_Distribution_Report

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Receiving_Agent	529	1-8	1	—	—
Distribution_Queue_ID	505	—	1	1	482
Distribution_ID	504	—	0-1 [1]	4-5	478
Queue_Entry_ID	528	32	0-1 [2]	—	—

Table 13 (Page 2 of 2). Receive_Distribution_Report

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Returned Parameter Name					
Return_Code [3]	540	ENUM	1	—	—
Report-To_Agent	531	1-8	1	—	—
Distribution_ID	504	—	0-1 [1]	4-5	478
Service_Parms	547	—	1	6	485
Reporting_DSU	537	—	1	2	—
Reporting_RGN	538	1-8	1	—	—
Reporting_REN	538	1-8	1	—	—
Report_Date	533	—	1	3	476
Report_Time	534	—	1	5	487
Agent_Correl	489	1-128	0-1	—	—
Report-To_DSU	532	—	1	2	—
Report-To_RGN	533	1-8	1	—	—
Report-To_REN	532	1-8	1	—	—
Report-To_User	533	—	0-1	2	—
Report-To_DGN	532	1-8	1	—	—
Report-To_DEN	531	1-8	1	—	—
Receiving_DSU	530	—	0-1	2	—
Receiving_RGN	530	1-8	1	—	—
Receiving_REN	530	1-8	1	—	—
Previously_Received_Date_Time	525	—	0-1	2	—
Previously_Received_Date	524	—	1	3	476
Previously_Received_Time	525	—	1	5	487
Queue_Entry_ID	528	32	0-1 [2]	—	—
Reported-On_Time	537	—	1	5	487
Reported-On_Dest_Agent	534	1-8	0-1*	—	—
SNA_Condition_Report	548	—	1	1-4	486
Integrity	509	ENUM	1	—	—

Notes:

1. When *distribution_ID* is used as a supplied parameter, it is not returned.
2. When *queue_entry_ID* is used as a supplied parameter, it is not returned.
3. If the value of *return_code* is not OK, then all other expected parameters are not returned.

VERB: Receiving_Sequence_Completed

Receiving_Sequence_Completed completes the transfer of responsibility from the DSU to the agent on a receiving sequence. This verb can be used to follow up either a Receive_Distribution or a Receive_Distribution_Report.

Table 14. Receiving_Sequence_Completed

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Distribution_Queue_ID	505	—	1	1	482
Queue_Entry_ID	528	32	1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—

VERB: Send_Distribution

Send_Distribution initiates a distribution within a DS network. In high-integrity verb sequences, the sequence number and date must be supplied. The methods of specifying destinations are:

1. The originating agent may choose to specify one or more users in the *destination* parameter. If so, the DSU determines for each user the DSU to which the distribution should be sent.
2. The originating agent might specify one or more DSUs as destinations. If so, the distribution service has no need to determine what DSU to send it to, either at the origin or as it travels through the network.

The originating agent may choose to specify a mix of (1) and (2).

Table 15 (Page 1 of 2). Send_Distribution

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Distribution_ID	504	—	1	4-5	478
Agent_Correl	489	1-128	0-1	—	—
Service_Parms	547	—	1	8	485
Exception_Report_Req	506	ENUM	1	—	—
Report-To_DSU	532	—	0-1	2	—
Report-To_RGN	533	1-8	1	—	—
Report-To_REN	532	1-8	1	—	—
Report-To_User	533	—	0-1	2	—
Report-To_DGN	532	1-8	1	—	—
Report-To_DEN	531	1-8	1	—	—
Report-To_Agent	531	1-8	0-1	—	—
Report_Service_Parms	534	—	0-1	6	482
Dest_Agent	500	1-8	0-1	—	—
Destination	503	—	1-256	1	476
Agent_Object	490	1-32763	0-1	—	—
Server	545	1-8	0-1	—	—
Server_Access	545	1-64	0-1*	—	—
Specific_Server_Info	549	1-32767	0-1*	—	—
Server_Object_Byte_Count	547	8	0-1	—	—
Integrity	509	ENUM	1	—	—

Table 15 (Page 2 of 2). Send_Distribution					
Parameter Name	**Parm Ref Page**	**Length**	**Occurrences**	**Children**	
				Num	**Subtab**
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—
Seqno_To_Clean_Up	544	—	0-1	3	—
Clean_Up_Seqno	494	4	1	—	—
Clean_Up_Date	494	—	1	3	476
Sending_State	544	ENUM	1	—	—
Sending_State	544	ENUM	1	—	—
Specific_Server_Report	549	1-32767	0-1	—	—
Distribution_Time	505	—	1	5	487

VERB: Sending_Sequence_Completed

Sending_Sequence_Completed is used only on high-integrity sequences after the agent has learned that the DSU has accepted responsibility for the distribution. After Sending_Sequence_Completed has been issued, the DSU may then purge its awareness of the distribution when it has completed sending.

Table 16. Sending_Sequence_Completed					
Parameter Name	**Parm Ref Page**	**Length**	**Occurrences**	**Children**	
				Num	**Subtab**
Supplied Parameter Name					
Distribution_ID	504	—	1	4-5	478
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—

Operations Verbs

VERB: Add_DSU_Data

Add_DSU_Data adds a new row to a system data structure.

Table 17. Add_DSU_Data

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Data_Structure	498	ENUM	1	—	—
New_Row	517	—	1	1	—
Agent_List_Entry	489	—	0-1	2-3	474
Connection_Definitions_Entry	496	—	0-1	14	475
Directory_Entry	504	—	0-1	3-6	477
DSU_Definition_Entry	505	—	0-1	7	479
Intervention_List_Entry	509	—	0-1	2	479
MU_ID_Registry_Entry	515	—	0-1	6-8	480
Next-DSU_Queue_Definitions_Entry	518	—	0-1	4-5	481
Routing_Table_Entry	542	—	0-1	5-14	484
Server_List_Entry	546	—	0-1	3-4	485
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—

VERB: Get_Distribution_Info

Get_Distribution_Info gives all the distribution control information for a specified distribution. The specified distribution may be in any of the DSU's queues.

Table 18. Get_Distribution_Info

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Queue_ID	529	—	1	1	482
Queue_Entry_ID	528	32	1	—	—
Returned Parameter Name					
Return_Code [1]	540	ENUM	1	—	—
Distribution_ID	504	—	1	4-5	478
Agent_Correl	489	1-128	0-1	—	—
Service_Parms	547	—	1	8	485
Destination	503	—	1-256	1	476
Dest_Agent	500	1-8	0-1	—	—
Agent_Object	490	1-32763	0-1	—	—
Server	545	1-8	0-1	—	—
Server_Access	545	1-64	0-1*	—	—
Server_Object_Byte_Count	547	8	0-1	—	—
Specific_Server_Info	549	1-32767	0-1*	—	—
Exception_Report_Req	506	ENUM	1	—	—
Report-To_DSU	532	—	0-1	2	—
Report-To_RGN	533	1-8	1	—	—
Report-To_REN	532	1-8	1	—	—
Report-To_User	533	—	0-1	2	—
Report-To_DGN	532	1-8	1	—	—
Report-To_DEN	531	1-8	1	—	—
Report-To_Agent	531	1-8	0-1	—	—
Report_Service_Parms	534	—	0-1	6	482
Hop_Count	508	4	1	—	—
Specific_Server_Report	549	1-32767	0-1	—	—
Previously_Received_Date_Time	525	—	0-1	2	—
Previously_Received_Date	524	—	1	3	476
Previously_Received_Time	525	—	1	5	487
Distribution_Time	505	—	1	5	487
Integrity	509	ENUM	1	—	—
MU_ID	515	4	0-1	—	—

Note:

1. If the value of *return_code* is not OK, then all other expected parameters are not returned.

VERB: Get_Distribution_Log_Entry

Get_Distribution_Log_Entry obtains a message (DTMU) recorded by a DSU in the distribution log.

Table 19. Get_Distribution_Log_Entry

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Distribution_ID	504	—	0-1	1-5	478
Requested_Entry_Number	538	4	1	—	—
Returned Parameter Name					
Return_Code [1]	540	ENUM	1	—	—
Number_Of_Matching_Entries	519	4	1	—	—
Logging_Date	511	—	1	3	476
Logging_Time	512	—	1	5	487
Program_Name	527	1-8	1	—	—
Session_Reference	547	—	0-1	4	—
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	1	—	—
Direction	503	ENUM	1	—	—
Accessed_Queue_ID	487	—	0-1	1	482
Product_Specific_Data	526	1-256	0-1	—	—
Distribution_ID	504	—	1	4-5	478
MU_ID	515	4	0-1	—	—
Destination	503	—	1-256	1	476
Distribution_Log_Data	505	1-32767	0-1	—	—

Note:

1. If the value of *return_code* is not OK, then all other expected parameters are not returned.

VERB: Get_Exception_Log_Entry

Get_Exception_Log_Entry returns the exception information recorded in the exception log.

Table 20. Get_Exception_Log_Entry

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Distribution_ID	504	—	0-1	1-5	478
Requested_Entry_Number	538	4	1	—	—
Returned Parameter Name					
Return_Code [1]	540	ENUM	1	—	—
Number_Of_Matching_Entries	519	4	1	—	—
Logging_Date	511	—	1	3	476
Logging_Time	512	—	1	5	487
Program_Name	527	1-8	1	—	—
Session_Reference	547	—	0-1	4	—
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	1	—	—
Direction	503	ENUM	1	—	—
Accessed_Queue_ID	487	—	0-1	1	482
Product_Specific_Data	526	1-256	0-1	—	—
Distribution_ID	504	—	0-1	2-5	478
SNA_Report_Code	548	4	1	—	—
MU_ID	515	4	0-1	—	—
Reported-On_Destination	537	—	≥0	1-2	483
Exception_Log_Data	506	1-32767	0-1	—	—

Note:

1. If the value of *return_code* is not OK, then all other expected parameters are not returned.

VERB: Hold_Distribution_Copy

Hold_Distribution_Copy holds a specified distribution copy on a specified queue.

Table 21. Hold_Distribution_Copy

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Queue_ID	529	—	1	1	482
Queue_Entry_ID	528	32	1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—

VERB: List_Adjacent_DSUs

List_Adjacent_DSUs lists the names of all DSUs that are shown in the routing table as being adjacent to the specified DSU.

Table 22. List_Adjacent_DSUs

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Service_Parms	547	—	0-1	2-8	485
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—
Adjacent_DSU	487	—	0-64	2	—
Adjacent_RGN	488	1-8	1	—	—
Adjacent_REN	488	1-8	1	—	—

VERB: List_Connections

List_Connections returns the connection names and their send/hold states for one or more connections. Connections can be specified individually by LU and mode, or collectively for all connections to an adjacent DSU. Connections can also be specified by a destination DSU, optionally qualified by service parameters.

Table 23. List_Connections

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Connection	496	—	0-64 [1]	2-3	—
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	0-1	—	—
Route	542	—	0-1 [1]	1-2	—
Dest_DSU	501	—	1	2	—
Dest_RGN	502	1-8	1	—	—
Dest_REN	502	1-8	1	—	—
Service_Parms	547	—	0-1	2-8	485
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—
Connection	496	—	0-64	4	—
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	1	—	—
Hold_State	507	ENUM	1	—	—

Note:

1. The *connection* and *route* supplied parameters are mutually exclusive.

VERB: List_Control_MU_Queue

List_Control_MU_Queue lists all control MUs present in a specified queue.

Table 24. List_Control_MU_Queue

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	1	—	—
Direction	503	ENUM	1	—	—
Starting_Queue_Entry_ID	549	32	0-1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—
Control_Info	497	—	≥0	2-3	—
MU_Type	516	ENUM	1	—	—
MU_ID	515	4	0-1	—	—
Queue_Entry_ID	528	32	1	—	—
Next_Queue_Entry_ID	518	32	0-1	—	—

VERB: List_Conversations

List_Conversations lists the active conversations, both sending and receiving, for a specified connection or group of connections.

Table 25. List_Conversations

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Connection	496	—	0-1	2-3	—
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	0-1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—
Conversation_Info	497	—	≥0	5	—
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	1	—	—
Conversation_ID	497	4	1	—	—
Direction	503	ENUM	1	—	—

VERB: List_Distributions_Being_Received

List_Distributions_Being_Received lists the distribution control information for distributions being received on a specified connection. Optionally, the verb can request the number of bytes already received by the server.

Table 26. List_Distributions_Being_Received

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Connection	496	—	0-1	3	—
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	1	—	—
Server_Bytes_Received	545	ENUM	1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—
Distribution_Info	504	—	≥0	2-3	—
Distribution_ID	504	—	1	4-5	478
Distribution_Time	505	—	1	5	487
Received_Server_Bytes	529	8	0-1	—	—

VERB: List_Distributions_Being_Sent

List_Distributions_Being_Sent lists the distribution control information for distributions being sent on a specified connection. Optionally, the information can be expanded to report the number of server object bytes yet to be sent.

Table 27. List_Distributions_Being_Sent

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Connection	496	—	0-1	3	—
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	1	—	—
Server_Bytes_Remaining	546	ENUM	1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—
Distribution_Info	504	—	≥0	2-3	—
Distribution_ID	504	—	1	4-5	478
Distribution_Time	505	—	1	5	487
Remaining_Server_Bytes	530	8	0-1	—	—

VERB: List_DSU_Data

List_DSU_Data is used to display to the operator the value of one or more rows of a system data structure.

Table 28. List_DSU_Data

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Data_Structure	498	ENUM	1	—	—
Row_Selection_Criteria	542	—	0-1	1	—
Agent_List_Entry	489	—	0-1	1-3	474
Connection_Definitions_Entry	496	—	0-1	1-14	475
Directory_Entry	504	—	0-1	1-6	477
DSU_Definition_Entry	505	—	0-1	1-7	479
Intervention_List_Entry	509	—	0-1	1-2	479
MU_ID_Registry_Entry	515	—	0-1	1-8	480
Next-DSU_Queue_Definitions_Entry	518	—	0-1	1-5	481
Routing_Table_Entry	542	—	0-1	1-14	484
Server_List_Entry	546	—	0-1	1-4	485
Column_To_Be_Listed	494	ENUM	0-14	—	—
Starting_Row_Number	549	4	0-1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—
Selected_Row	544	—	0-64	1	—
Agent_List_Entry	489	—	0-1	1-3	474
Connection_Definitions_Entry	496	—	0-1	1-14	475
Directory_Entry	504	—	0-1	1-6	477
DSU_Definition_Entry	505	—	0-1	1-7	479
Intervention_List_Entry	509	—	0-1	1-2	479
MU_ID_Registry_Entry	515	—	0-1	1-8	480
Next-DSU_Queue_Definitions_Entry	518	—	0-1	1-5	481
Routing_Table_Entry	542	—	0-1	1-14	484
Server_List_Entry	546	—	0-1	1-4	485
New_Row_Number	517	4	0-1	—	—

VERB: List_Queue_Entries

List_Queue_Entries lists the distributions, the distribution reports, and server reports for a specified queue.

Table 29. List_Queue_Entries

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Queue_ID	529	—	1	1	482
Queue_Entry_Type	528	ENUM	0-1	—	—
Distribution_ID	504	—	0-1	1-5	478
Querying_Agent	528	1-8	1	—	—
After_Date	488	—	0-1*	3	476
After_Time	488	—	0-1*	5	487
Before_Date	492	—	0-1*	3	476
Before_Time	492	—	0-1*	5	487
Starting_Queue_Entry_ID	549	32	0-1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—
Distribution_Info	504	—	≥0	3-6	—
Distribution_ID	504	—	0-1	4-5	478
Queue_Entry_ID	528	32	1	—	—
Queue_Entry_Type	528	ENUM	1	—	—
Distribution_Time	505	—	0-1	5	487
Previously_Received_Date_Time	525	—	0-1	2	—
Previously_Received_Date	524	—	1	3	476
Previously_Received_Time	525	—	1	5	487
Hold_State	507	ENUM	1	—	—
Next_Queue_Entry_ID	518	32	0-1	—	—

VERB: List_Queues_Containing_Distribution

List_Queues_Containing_Distribution lists the queue identifiers of all queues that contain a copy of the specified distribution.

Table 30. List_Queues_Containing_Distribution

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Distribution_ID	504	—	1	4-5	478
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—
Queue_ID	529	—	0-64	1	482

Note:

1. If the value of *return_code* is not OK, then all other expected parameters are not returned.

VERB: Modify_DSU_Data

Modify_DSU_Data changes the value of one or more rows in a system data structure. This verb must be used for parameters that must always exist, such as the DSU name. It may also be used to modify an existing list entry.

Table 31. Modify_DSU_Data

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Data_Structure	498	ENUM	1	—	—
Row_Selection_Criteria	542	—	1	1	—
Agent_List_Entry	489	—	0-1	1-3	474
Connection_Definitions_Entry	496	—	0-1	1-14	475
Directory_Entry	504	—	0-1	1-6	477
DSU_Definition_Entry	505	—	0-1	1-7	479
Intervention_List_Entry	509	—	0-1	1-2	479
MU_ID_Registry_Entry	515	—	0-1	1-8	480
Next-DSU_Queue_Definitions_Entry	518	—	0-1	1-5	481
Routing_Table_Entry	542	—	0-1	1-14	484
Server_List_Entry	546	—	0-1	1-4	485
Modified_Row	514	—	1	1	—
Agent_List_Entry	489	—	0-1	2-3	474
Connection_Definitions_Entry	496	—	0-1	14	475
Directory_Entry	504	—	0-1	3-6	477
DSU_Definition_Entry	505	—	0-1	7	479
Intervention_List_Entry	509	—	0-1	2	479
MU_ID_Registry_Entry	515	—	0-1	6-8	480
Next-DSU_Queue_Definitions_Entry	518	—	0-1	4-5	481
Routing_Table_Entry	542	—	0-1	5-14	484
Server_List_Entry	546	—	0-1	3-4	485
If_Nonunique_Key	508	ENUM	1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—

VERB: Purge_Queue_Entry

Purge_Queue_Entry deletes a queue entry from a specified queue and causes appropriate reporting.

Table 32. Purge_Queue_Entry

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Queue_ID	529	—	1	1	482
Queue_Entry_ID	528	32	1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—

VERB: Release_Distribution_Copy

Release_Distribution_Copy releases a specified distribution copy on a specified queue.

Table 33. Release_Distribution_Copy

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Queue_ID	529	—	1	1	482
Queue_Entry_ID	528	32	1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—

VERB: Remove_DSU_Data

Remove_DSU_Data removes one or more rows from a system data structure.

Table 34. Remove_DSU_Data

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Data_Structure	498	ENUM	1	—	—
Row_Selection_Criteria	542	—	1	1	—
Agent_List_Entry	489	—	0-1	1-3	474
Connection_Definitions_Entry	496	—	0-1	1-14	475
Directory_Entry	504	—	0-1	1-6	477
DSU_Definition_Entry	505	—	0-1	1-7	479
Intervention_List_Entry	509	—	0-1	1-2	479
MU_ID_Registry_Entry	515	—	0-1	1-8	480
Next-DSU_Queue_Definitions_Entry	518	—	0-1	1-5	481
Routing_Table_Entry	542	—	0-1	1-14	484
Server_List_Entry	546	—	0-1	1-4	485
If_Nonunique_Key	508	ENUM	1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—

VERB: Reroute_Distribution_Copies

Reroute_Distribution_Copies reroutes one or more distribution copies found in a specified queue.

Table 35. Reroute_Distribution_Copies

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Connection	496	—	0-64	2-3	—
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	0-1	—	—
Distribution_ID	504	—	0-1	4-5	478
Service_Parms	547	—	0-1	2-8	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—

VERB: Reset_MU_ID_Registry

Reset_MU_ID_Registry causes the MU_ID registry for a connection to be resynchronized. Issuing this verb causes a Reset Request MU to be sent on the specified connection.

Table 36. Reset_MU_ID_Registry

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Connection	496	—	1	3	—
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	1	—	—
Next_MU_ID	518	4	1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—

VERB: Start_Connection

Start_Connection activates a connection by causing the persistent sessions, if any, to be bound, setting the DS_Send control values, and checking that the appropriate routing segments are in the active state.

Table 37. Start_Connection

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Connection	496	—	1	3	—
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	1	—	—
Max_DS_Sends	513	4	1	—	—
Max_DS_Receives	512	4	1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—

VERB: Terminate_Connection

Terminate_Connection terminates a connection by terminating all active conversations on it.

Table 38. Terminate_Connection

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Connection	496	—	1	3	—
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	1	—	—
MU_Action	514	ENUM	1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—

VERB: Terminate_Conversation

Terminate_Conversation terminates a conversation.

Table 39. Terminate_Conversation

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Conversation_ID	497	4	1	—	—
MU_Action	514	ENUM	1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—

Server Verbs

VERB: Assign_Read_Access

Assign_Read_Access creates entries in the access list. Any process that has been assigned read access to a particular server object can issue this verb. The issuer of the verb is identified in each entry as the assigning process. When the assigning and the assigned-to processes are the same, the entry is called a *self assignment.* When they are different, the entry is called an *other assignment.* Typically, a process will issue this as soon as it has been given an access descriptor to confirm access and to create its own self-assignment entry. DS does this before returning control on a Send_Distribution verb.

Every time this verb is issued, an entry is created and it is the responsibility of the assigning process to eventually issue a Release_Read_Access to delete the entry. The access list may contain multiple entries with the same contents.

Table 40. Assign_Read_Access

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Assigning_Process	491	1-8	1	—	—
Current_Unit_Of_Work_ID	498	—	1	1	—
Distribution_ID	504	—	0-1	4-5	478
Agent_Unit_Of_Work_ID	490	1-256	0-1	—	—
Assigned_Process	490	1-8	1	—	—
New_Unit_Of_Work_ID	518	—	0-1	1	—
Distribution_ID	504	—	0-1	4-5	478
Agent_Unit_Of_Work_ID	490	1-256	0-1	—	—
Assign_Unit_Of_Work_ID	490	ENUM	1	—	—
Server	545	1-8	1	—	—
Server_Access	545	1-64	1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—

VERB: Backout_Server_Object

Backout_Server_Object Informs the server that it should back out a server object that has been successfully written, because the distribution will not be delivered.

Table 41. Backout_Server_Object

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Requesting_Process	538	1-8	1	—	—
Server	545	1-8	1	—	—
Server_Access	545	1-64	0-1*	—	—
Specific_Server_Report	549	1-32767	0-1	—	—
Specific_Server_Info	549	1-32767	0-1*	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—
Specific_Server_Report	549	1-32767	0-1	—	—

VERB: Initiate_Read

Initiate_Read creates and initializes the control block used during the reading of the server object.

Table 42. Initiate_Read

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Requesting_Process	538	1-8	1	—	—
Server	545	1-8	1	—	—
Unit_Of_Work_ID	554	—	1	1	—
Distribution_ID	504	—	0-1	2-5	478
Agent_Unit_Of_Work_ID	490	1-256	0-1	—	—
Agent_Correl	489	1-128	0-1	—	—
Agent_Object	490	1-32763	0-1	—	—
Specific_Server_Info	549	1-32767	0-1*	—	—
Server_Access	545	1-64	0-1*	—	—
Restartability	540	ENUM	1	—	—
Restart_ID	539	—	0-1	4	—
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	1	—	—
MU_ID	515	4	1	—	—
Restart_Byte	539	8	0-1	—	—
Returned Parameter Name					
Return_Code [1]	540	ENUM	1	—	—
Server_Instance_ID	546	4	1	—	—
Specific_Server_Report	549	1-32767	0-1	—	—

Note:

1. If the value of *return_code* is not OK, then all other expected parameters are not returned, except *specific_server_report* may still be returned.

VERB: Initiate_Write

Initiate_Write creates and initializes the control block used during the writing of the server object. An exclusive-write lock is set to indicate that no reading, other writing, or deletion may take place.

As long as the exclusive-write lock for an object is set, no read access can be assigned. Thus, any Assign_Read_Access issued while the exclusive-write lock is set results in an error condition.

Table 43. Initiate_Write

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Requesting_Process	538	1-8	1	—	—
Server	545	1-8	1	—	—
Server_Object_Byte_Count	547	8	0-1	—	—
Unit_Of_Work_ID	554	—	1	1	—
Distribution_ID	504	—	0-1	4-5	478
Agent_Unit_Of_Work_ID	490	1-256	0-1	—	—
Agent_Correl	489	1-128	0-1	—	—
Agent_Object	490	1-32763	0-1	—	—
Restartability	540	ENUM	1	—	—
Restart_ID	539	—	0-1	4	—
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	1	—	—
MU_ID	515	4	1	—	—
Restart_Byte	539	8	0-1	—	—
Returned Parameter Name					
Return_Code [1]	540	ENUM	1	—	—
Server_Instance_ID	546	4	1	—	—
Specific_Server_Report	549	1-32767	0-1	—	—

Note:

1. If the value of *return_code* is not OK, then all other expected parameters are not returned, except *specific_server_report* may still be returned.

VERB: Query_Last_Byte_Received

Query_Last_Byte_Received returns the last byte that was successfully written by the server.

Table 44. Query_Last_Byte_Received

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Requesting_Process	538	1-8	1	—	—
Server	545	1-8	1	—	—
Restart_ID	539	—	1	4	—
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	1	—	—
MU_ID	515	4	1	—	—
Returned Parameter Name					
Return_Code [1]	540	ENUM	1	—	—
Last_Byte_Received	510	8	1	—	—

Note:

1. If the value of *return_code* is not OK, then all other expected parameters are not returned.

VERB: Read

Read returns a contiguous stream of bytes as a part of the server object.

Table 45. Read

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Server	545	1-8	1	—	—
Server_Instance_ID	546	4	1	—	—
Buffer_Length	493	8	1	—	—
Buffer	492	1-32767	1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—
Data_Length	498	8	1	—	—

VERB: Release_Read_Access

Release_Read_Access is issued by an assigning process to cause an entry to be deleted from the access list. It must be issued once for every Assign_Read_Access that the assigning process has issued, including the Assign_Read_Access implied by the Terminate_Write verb. Typically, a process will delete its self-assignment entry only after assigning read access to another process. It will delete its other-assignment entries only after the assigned-to process has had an opportunity to create its self-assignment entry. When all access list entries are deleted, the server will delete the object.

Table 46. Release_Read_Access

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Assigning_Process	491	1-8	1	—	—
Unit_Of_Work_ID	554	—	1	1	—
Distribution_ID	504	—	0-1	4-5	478
Agent_Unit_Of_Work_ID	490	1-256	0-1	—	—
Assigned_Process	490	1-8	1	—	—
Server	545	1-8	1	—	—
Server_Access	545	1-64	1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—

VERB: Terminate_Read

Terminate_Read deallocates the control block used during the reading of the server object.

Table 47. Terminate_Read

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Server	545	1-8	1	—	—
Server_Instance_ID	546	4	1	—	—
Termination_Type	554	ENUM	1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—
Specific_Server_Report	549	1-32767	0-1	—	—

VERB: Terminate_Restartability

Terminate_Restartability informs the server that the identified server object will not be restarted. (That is, no Initiate_Write or Initiate_Read verbs using the *restart_byte* parameter will be issued.)

Table 48. Terminate_Restartability

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Requesting_Process	538	1-8	1	—	—
Server	545	1-8	1	—	—
Restart_ID	539	—	1	4	—
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	1	—	—
MU_ID	515	4	1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—
Specific_Server_Report	549	1-32767	0-1	—	—

VERB: Terminate_Write

Terminate_Write deallocates the control block used for the writing of the server object, releases the exclusive-write lock, and generates the first entry in the read access list. This entry is a self-assignment entry for the requesting process that issued the Initiate_Write and contains the unit of work specified on that verb, if any.

Table 49. Terminate_Write

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Server	545	1-8	1	—	—
Server_Instance_ID	546	4	1	—	—
Termination_Type	554	ENUM	1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—
Specific_Server_Report	549	1-32767	0-1	—	—
Server_Access	545	1-64	0-1*	—	—
Specific_Server_Info	549	1-32767	0-1*	—	—

VERB: Write

Write passes to the server a part of a server object.

Table 50. Write

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Supplied Parameter Name					
Server	545	1-8	1	—	—
Server_Instance_ID	546	4	1	—	—
Buffer	492	1-32767	1	—	—
Data_Length	498	8	1	—	—
Returned Parameter Name					
Return_Code	540	ENUM	1	—	—

Subtables

SUBTABLE: Agent_List_Entry

Table 51. Agent_List_Entry

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Agent_List_Entry [1]	489	—	—	2-3 [2]	—
Agent	489	1-8	1	—	—
Local_Queue_Type	511	ENUM	1	—	—
Local_Info	510	1-64	0-1	—	—

Notes:

1. The *agent_list_entry* parameter is supplied on Add_DSU_Data, List_DSU_Data, Modify_DSU_Data, and Remove_DSU_Data.

   It is returned on List_DSU_Data.

2. When *agent_list_entry* occurs as a child of *row_selection_criteria* or *selected_row*, any children of *agent_list_entry* may be used for selection, therefore nullifying the occurrences column.

SUBTABLE: Connection_Definitions_Entry

Table 52. Connection_Definitions_Entry

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Connection_Definitions_Entry [1]	496	—	—	14 [2]	—
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	1	—	—
Direction	503	ENUM	1	—	—
Max_DS_Sends	513	4	0-1*	—	—
Max_DS_Receives	512	4	0-1*	—	—
Data_Stream_Format	498	ENUM	1	—	—
Year [3]	554	4	1	—	—
Month [3]	514	4	1	—	—
Day [3]	499	4	1	—	—
Hours [3]	508	4	1	—	—
Minutes [3]	513	4	1	—	—
Seconds [3]	543	4	1	—	—
Hundredths [3]	508	4	1	—	—
Local_Or_GMT_Flag [3]	510	ENUM	1	—	—

Notes:

1. The *connection_definitions_entry* parameter is supplied on Add_DSU_Data, List_DSU_Data, Modify_DSU_Data, and Remove_DSU_Data.

   It is returned on List_DSU_Data.

2. When *connection_definitions_entry* occurs as a child of *row_selection_criteria* or *selected_row*, any children of *connection_definitions_entry* may be used for selection, therefore nullifying the occurrences column.

3. The date and time parameters indicate the last time the MU_ID Registry was reset for this connection. This is the date and time that appeared in the RRMU.

SUBTABLE: Date

Table 53. Date

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Date [1]	—	—	—	3	—
Year	554	4	1	—	—
Month	514	4	1	—	—
Day	499	4	1	—	—

Note:

1. A date is supplied, as *after_date* and *before_date*, on List_Queue_Entries.

   It is returned on Get_Distribution_Info (*previously_received_date*), Get_Distribution_Log_Entry (*logging_date*), Get_Exception_Log_Entry (*logging_date*), List_Queue_Entries (*previously_received_date*), Receive_Distribution (*previously_received_date*), Receive_Distribution_Report (*report_date*, *previously_received_date*), and Send_Distribution (*clean_up_date*).

   It is also found in the *distribution_identification* subtable as *origin_date.*

SUBTABLE: Destination

Table 54. Destination

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Destination [1]	503	—	—	1-2*	—
Dest_DSU	501	—	0-1	2	—
Dest_RGN	502	1-8	1	—	—
Dest_REN	502	1-8	1	—	—
Dest_User	502	—	0-1	2	—
Dest_DGN	501	1-8	1	—	—
Dest_DEN	500	1-8	1	—	—

Note:

1. The *destination* parameter is supplied on Send_Distribution.

   It is returned on Get_Distribution_Info, Get_Distribution_Log_Entry, and Receive_Distribution.

SUBTABLE: Directory_Entry

Table 55. Directory_Entry

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Directory_Entry [1]	504	—	—	3-6 [2]	—
DGN	503	1-8	0-1 [3]	—	—
DEN	499	1-8	0-1 [3]	—	—
Agent	489	1-8	1 [3]	—	—
RGN	541	1-8	0-1 [4]	—	—
REN	531	1-8	0-1 [4]	—	—
Local_Queue_Type	511	ENUM	0-1 [4]	—	—
Local_Info	510	1-64	0-1 [4]	—	—
Next_Seqno	519	4	0-1 [4]	—	—

Notes:

1. The *directory_entry* parameter is supplied on Add_DSU_Data, List_DSU_Data, Modify_DSU_Data, and Remove_DSU_Data.

   It is returned on List_DSU_Data.

2. When *directory_entry* occurs as a child of *row_selection_criteria* or *selected_row*, any children of *directory_entry* may be used for selection, therefore nullifying the occurrences column.

3. The *DGN* and *DEN* must be specified as a pair. The *agent*, the *DEN*, or the *DGN* and *DEN* may be specified with a system-specific value that matches any token for default directing.

4. A given entry contains either *RGN* and *REN* or *local_queue_type* and *next_seqno*; however, an entry never contains both pairs of parameters. An entry that contains *local_queue_type* and *next_seqno* may also contain *local_info*.

SUBTABLE: Distribution_ID

Table 56. Distribution_ID

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Distribution_ID [1]	504	—	—	4-5 [2]	—
Origin_DSU	521	—	0-1 [3]	2	—
Origin_RGN	521	1-8	1	—	—
Origin_REN	521	1-8	1	—	—
Origin_User	522	—	0-1	2	—
Origin_DGN	520	1-8	1	—	—
Origin_DEN	520	1-8	1	—	—
Origin_Agent	519	1-8	1	—	—
Origin_Seqno	522	4	1	—	—
Origin_Date	520	—	1	3	476

Notes:

1. The *distribution_identification* parameter is supplied on Assign_Read_Access, Get_Distribution_Log_Entry, Get_Exception_Log_Entry, Initiate_Read, Initiate_Write, List_Queue_Entries, List_Queues_Containing_Distribution, Query_Distribution_Sending, Receive_Distribution, Receive_Distribution_Report, Release_Read_Access, Reroute_Distribution_Copies, Send_Distribution, and Sending_Sequence_Completed.

   It is returned on Get_Distribution_Info, Get_Distribution_Log_Entry, Get_Exception_Log_Entry, List_Distributions_Being_Received, List_Distributions_Being_Sent, List_Queue_Entries, Receive_Distribution, and Receive_Distribution_Report.

2. The number of children of *distribution_identification* is shown on some verbs as 1-5. On these verbs, any of the children of *distribution_identification* may be used as selection criteria.

3. The *origin_DSU* will always be present in a *distribution_identification* except in two cases:

   - The *distribution_identification* is being used as a selection criteria, as described in Note 2.
   - The *distribution_identification* was transported in an FS1 Dist_MU *type* REPORT. In this case, *origin_user* will always be present.

SUBTABLE: DSU_Definition_Entry

Table 57. DSU_Definition_Entry

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
DSU_Definition_Entry [1]	505	—	—	7 [2]	—
RGN	541	1-8	1	—	—
REN	531	1-8	1	—	—
Default_Hop_Count	499	2	1	—	—
Logging	511	ENUM	1	—	—
GMT_Offset_Hours	507	4	1	—	—
GMT_Offset_Minutes	507	4	1	—	—
GMT_Offset_Direction	506	ENUM	1	—	—

Notes:

1. The *DSU_definition_entry* parameter is supplied on Add_DSU_Data, List_DSU_Data, Modify_DSU_Data, and Remove_DSU_Data.

   It is returned on List_DSU_Data.

2. When *DSU_definition_entry* occurs as a child of *row_selection_criteria* or *selected_row*, any children of *DSU_definition_entry* may be used for selection, therefore nullifying the occurrences column.

SUBTABLE: Intervention_List_Entry

Table 58. Intervention_List_Entry

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Intervention_List_Entry [1]	509	—	—	2 [2]	—
RGN	541	1-8	1	—	—
REN	531	1-8	1	—	—

Notes:

1. The *intervention_list_entry* parameter is supplied on Add_DSU_Data, List_DSU_Data, Modify_DSU_Data, and Remove_DSU_Data.

   It is returned on List_DSU_Data.

2. When *intervention_list_entry* occurs as a child of *row_selection_criteria* or *selected_row*, any children of *intervention_list_entry* may be used for selection, therefore nullifying the occurrences column.

SUBTABLE: MU_ID_Registry_Entry

Table 59. MU_ID_Registry_Entry

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
MU_ID_Registry_Entry [1]	515	—	—	6-8 [2]	—
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	1	—	—
Direction	503	ENUM	1	—	—
MU_ID	515	4	1	—	—
MU_ID_State	515	ENUM	1	—	—
MU_Instance_Number	516	4	0-1	—	—
Queue_Entry_ID	528	4	0-1	—	—

Notes:

1\.

a. The *MU_ID_registry_entry* parameter is supplied on Add_DSU_Data, List_DSU_Data, Modify_DSU_Data, and Remove_DSU_Data.

It is returned on List_DSU_Data.

b. The model of the protocol boundary assumes one MU_ID Registry for an entire DSU.

2. When *MU_ID_registry_entry* occurs as a child of *row_selection_criteria* or *selected_row*, any children of *MU_ID_registry_entry* may be used for selection, therefore nullifying the occurrences column.

SUBTABLE: Next-DSU_Queue_Definitions_Entry

Table 60. Next-DSU_Queue_Definitions_Entry

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Next-DSU_Queue_Definitions_Entry [1]	518	—	—	4-5 [2]	—
Scheduling_Data	542	1-256	0-1	—	—
Hold_State	507	ENUM	1	—	—
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	1	—	—

Notes:

1.
   a. The *next-DSU_queue_definitions_entry* parameter is supplied on Add_DSU_Data, List_DSU_Data, Modify_DSU_Data, and Remove_DSU_Data.

      It is returned on List_DSU_Data.

   b. The model of the protocol boundary assumes one next-DSU queue per connection.

2. When *next-DSU_queue_definitions_entry* occurs as a child of *row_selection_criteria* or *selected_row*, any children of *next-DSU_queue_definitions_entry* may be used for selection, therefore nullifying the occurrences column.

SUBTABLE: Queue_ID

Table 61. Queue_ID

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Queue_ID [1]	529	—	—	1	—
Connection	496	—	0-1	5	—
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	1	—	—
Direction	503	ENUM	1	—	—
Connection_Queue_Type	497	ENUM	1	—	—
Dest_User	502	—	0-1	2	—
Dest_DGN	501	1-8	1	—	—
Dest_DEN	500	1-8	1	—	—
Dest_Agent	500	1-8	0-1	—	—

Note:

1. The *queue_identifier* parameter is supplied, under various names, on several verbs. It is supplied on Get_Distribution_Info (*queue_ID*), Hold_Distribution_Copy (*queue_ID*), List_Queue_Entries (*queue_ID*), Obtain_Local_Server_Report (*distribution_queue_ID*), Purge_Queue_Entry (*queue_ID*), and Release_Distribution_Copy (*queue_ID*), Receive_Distribution (*distribution_queue_ID*), Receive_Distribution_Report (*distribution_queue_ID*), and Receiving_Sequence_Completed (*distribution_queue_ID*).

   It is returned on Get_Distribution_Log_Entry (*accessed_queue_ID*), Get_Exception_Log_Entry (*accessed_queue_ID*), and List_Queues_Containing_Distribution (*queue_ID*).

SUBTABLE: Report_Service_Parms

Table 62. Report_Service_Parms

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Report_Service_Parms [1]	534	—	—	6	—
Priority	525	ENUM	1	—	—
Priority_Comp_Op	526	ENUM	1	—	—
Protection	527	ENUM	1	—	—
Protection_Comp_Op	527	ENUM	1	—	—
Security	543	ENUM	1	—	—
Security_Comp_Op	543	ENUM	1	—	—

Note:

1. The *report_service_parameters* parameter is supplied on Send_Distribution.

   It is returned on Get_Distribution_Info and Receive_Distribution.

SUBTABLE: Reported-On_Destination

Table 63. Reported-On_Destination

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Reported-On_Destination [1]	537	—	—	1-2	—
Reported-On_Dest_DSU	536	—	0-1*	2	—
Reported-On_Dest_RGN	536	1-8	1	—	—
Reported-On_Dest_REN	536	1-8	1	—	—
Reported-On_Dest_User	537	—	0-1	2	—
Reported-On_Dest_DGN	535	1-8	1	—	—
Reported-On_Dest_DEN	535	1-8	1	—	—

Note:

1. The *reported-on_destination* parameter is returned on Get_Exception_Log_Entry. It is also found in the *SNA_condition_report* subtable.

SUBTABLE: Routing_Table_Entry

Table 64. Routing_Table_Entry

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Routing_Table_Entry [1]	542	—	—	5-14 [2]	—
Dest_RGN [4]	502	1-8	1	—	—
Dest_REN [4]	502	1-8	1	—	—
Priority	525	ENUM	0-1 [3]	—	—
Priority_Comp_Op	526	ENUM	0-1	—	—
Protection	527	ENUM	0-1 [3]	—	—
Protection_Comp_Op	527	ENUM	0-1	—	—
Capacity	493	ENUM	0-1 [3]	—	—
Capacity_Comp_Op	494	ENUM	0-1	—	—
Security	543	ENUM	0-1 [3]	—	—
Security_Comp_Op	543	ENUM	0-1	—	—
Originating_Hop_Count	522	4	0-1	—	—
Net_ID	517	1-8	1	—	—
LU_Name	512	1-8	1	—	—
Mode_Name	513	1-8	1	—	—

Notes:

1. The *routing_table_entry* parameter is supplied on Add_DSU_Data, List_DSU_Data, Modify_DSU_Data, and Remove_DSU_Data.

   It is returned on List_DSU_Data.

2. When *routing_table_entry* occurs as a child of *row_selection_criteria* or *selected_row*, any children of *routing_table_entry* may be used for selection, therefore nullifying the occurrences column.

3. The presence of each service parameter is dependent upon the presence of its corresponding comparison operator.

4. The *dest_REN*, or the *dest_RGN* and *dest_REN* may be specified with a system-specific value that matches any token for default routing.

SUBTABLE: Server_List_Entry

Table 65. Server_List_Entry

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Server_List_Entry [1]	546	—	—	3-4 [2]	—
Server	545	1-8	1	—	—
Local_Info	510	1-64	0-1	—	—
Restart_Capable	539	ENUM	1	—	—
Auxiliary_Operations	491	ENUM	1	—	—

Notes:

1. The *server_list_entry* parameter is supplied on Add_DSU_Data, List_DSU_Data, Modify_DSU_Data, and Remove_DSU_Data.

   It is returned on List_DSU_Data.

2. When *server_list_entry* occurs as a child of *row_selection_criteria* or *selected_row*, any children of *server_list_entry* may be used for selection, therefore nullifying the occurrences column.

SUBTABLE: Service_Parms

Table 66. Service_Parms

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Service_Parms [1]	547	—	—	8 [2]	—
Priority	525	ENUM	1	—	—
Priority_Comp_Op	526	ENUM	1	—	—
Protection	527	ENUM	1	—	—
Protection_Comp_Op	527	ENUM	1	—	—
Capacity	493	ENUM	1 [3]	—	—
Capacity_Comp_Op	494	ENUM	1 [3]	—	—
Security	543	ENUM	1	—	—
Security_Comp_Op	543	ENUM	1	—	—

Notes:

1. The *service_parameters* parameter is supplied on List_Adjacent_DSUs, List_Connections, Reroute_Distribution_Copies, and Send_Distribution.

   It is returned on Get_Distribution_Info, Receive_Distribution, and Receive_Distribution_Report.

2. When *service_parms* are returned on Receive_Distribution_Report, *capacity* and *capacity_comp_op* are not used.

3. When used as selection criteria, a minimum of one service parameter/comparison operator combination may be specified.

SUBTABLE: SNA_Condition_Report

Table 67. SNA_Condition_Report

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
SNA_Condition_Report [1]	548	—	—	1-4	—
SNA_Report_Code	548	4	1	—	—
Structure_Report	552	—	0-10*	2-7	—
Structure_State	553	4	1	—	—
Structure_Contents	551	1-98	0-1*	—	—
Parent_Spec	524	—	0-7	2-4	—
Parent_ID_Or_T	523	1-2	1	—	—
Parent_Class	523	ENUM	1	—	—
Parent_Position	524	2	0-1	—	—
Parent_Instance	523	2	0-1	—	—
Structure_Spec	553	—	0-1*	1-4	—
Structure_ID_Or_T	551	1-2	0-1*	—	—
Structure_Class	550	ENUM	1	—	—
Structure_Position	552	2	0-1	—	—
Structure_Instance	551	2	0-1	—	—
Structure_Segment_Num	552	2	0-1*	—	—
Structure_Byte_Offset	550	2	0-1	—	—
Sibling	548	1	0-98*	—	—
Reported-On_Destination	537	—	≥1	1-2	483
Supplemental_Report	553	1-253	0-5*	—	—

Note:

1. The *SNA_condition_report* parameter is returned on Receive_Distribution_Report.

SUBTABLE: Time

Table 68. Time

Parameter Name	Parm Ref Page	Length	Occurrences	Children	
				Num	Subtab
Time [1]	—	—	—	5	—
Hours	508	4	1	—	—
Minutes	513	4	1	—	—
Seconds	543	4	1	—	—
Hundredths	508	4	1	—	—
Local_Or_GMT_Flag	510	ENUM	1	—	—

Note:

1. A time is supplied, as *after_time* and *before_time*, on List_Queue_Entries.

   It is returned on Get_Distribution_Info (*previously_received_time*, *distribution_time*), Get_Distribution_Log_Entry (*logging_time*), Get_Exception_Log_Entry (*logging_time*), List_Distributions_Being_Received (*distribution_time*), List_Distributions_Being_Sent (*distribution_time*), List_Queue_Entries (*distribution_time*, *previously_received_time*), Receive_Distribution (*previously_received_time*, *distribution_time*), Receive_Distribution_Report (*report_time*, *previously_received_time*, *reported-on_time*), and Send_Distribution (*distribution_time*).

Parameter descriptions

Accessed_Queue_ID

Description: The *accessed_queue_identifier* is the queue containing the logged distribution, if any is applicable.

Verbs Supplied on: None

Verbs Returned on: Get_Distribution_Log_Entry, Get_Exception_Log_Entry

Adjacent_DSU

Description: An *adjacent_DSU* is a DSU with which this DSU has an active connection or a DSU with which this DSU routinely establishes a connection.

Verbs Supplied on: None

Verbs Returned on: List_Adjacent_DSUs

Adjacent_REN

Description: The *adjacent_REN* is the second part of the name of the adjacent DSU. This is typically, but not necessarily, the LU name.

Verbs Supplied on: None

Verbs Returned on: List_Adjacent_DSUs

Format: Character string

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

Adjacent_RGN

Description: The *adjacent_RGN* is the first part of the name of the adjacent DSU. This is typically, but not necessarily, the network ID.

Verbs Supplied on: None

Verbs Returned on: List_Adjacent_DSUs

Format: Character string

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

After_Date

Description: The *after_date* parameter, along with the *after_time* parameter, limits the selection of distributions to be listed to those that were originated after a certain date and time.

Verbs Supplied on: List_Queue_Entries

Verbs Returned on: None

Presence Rule: Occurs when *after_time* is present.

After_Time

Description: The *after_time* parameter, along with the *after_date* parameter, limits the selection of distributions to be listed to those that were originated after a certain date and time.

Verbs Supplied on: List_Queue_Entries

Verbs Returned on: None

Presence Rule: Occurs when *after_date* is present.

Agent

Description: The *agent* is an application transaction program that interacts with DS on behalf of a user or a DSU.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *agent_list_entry*, *directory_entry*

Format: Character string, except for first byte

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

The first byte of an SNA-registered transaction program name ranges in value from X'00' to X'3F'. When the first byte ranges in value from X'41' to X'FF', the transaction program is not SNA registered. X'40' is not a valid first byte.

Agent_Correl

Description: The *agent_correlation* is a string supplied by the origin agent. DS is not aware of its contents.

Verbs Supplied on: Initiate_Read, Initiate_Write, Send_Distribution

Verbs Returned on: Get_Distribution_Info, Receive_Distribution, Receive_Distribution_Report

Format: Undefined byte string

Agent_List_Entry

Description: The *agent_list_entry* describes one row in the Agent List data structure.

Verbs Supplied on: Add_DSU_Data, List_DSU_Data, Modify_DSU_Data, Remove_DSU_Data

Verbs Returned on: List_DSU_Data

Agent_Object

Description: The *agent_object* is directly supplied by the origin agent. It is never parsed by the distribution service and is directly delivered, unchanged, to the agent at each destination.

Verbs Supplied on: Initiate_Read, Initiate_Write, Send_Distribution

Verbs Returned on: Get_Distribution_Info, Receive_Distribution, Receive_Distribution_Report

Format: Undefined byte string

Agent_Unit_Of_Work_ID

Description: The *agent_unit_of_work_identifier* is an agent-defined byte string that the agent uses to identify a server unit of work.

Verbs Supplied on: Assign_Read_Access, Initiate_Read, Initiate_Write, Release_Read_Access

Verbs Returned on: None

Format: Undefined byte string

Assign_Unit_Of_Work_ID

Description: The *assign_unit_of_work_identifier* flag indicates whether a new unit-of-work identifier is being assigned to a server unit of work.

Verbs Supplied on: Assign_Read_Access

Verbs Returned on: None

Format: Enumeration

Value	Meaning
YES	A new unit of work is being used when assigning access to a process.
NO	The old unit of work is being used when assigning access to a process.

Assigned_Process

Description: The *assigned_process* is the process that is being assigned read access to a server object.

Verbs Supplied on: Assign_Read_Access, Release_Read_Access

Verbs Returned on: None

Format: Character string, except for first byte

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X′40′) characters are not allowed.

The first byte of an SNA-registered transaction program name ranges in value from X′00′ to X′3F′. When the first byte ranges in value from X′41′ to X′FF′, the transaction program is not SNA registered. X′40′ is not a valid first byte.

Assigning_Process

Description: The *assigning_process* is the process that is assigning read access to a server object, either to another process or to itself.

Verbs Supplied on: Assign_Read_Access, Release_Read_Access

Verbs Returned on: None

Format: Character string, except for first byte

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X′40′) characters are not allowed.

The first byte of an SNA-registered transaction program name ranges in value from X′00′ to X′3F′. When the first byte ranges in value from X′41′ to X′FF′, the transaction program is not SNA registered. X′40′ is not a valid first byte.

Auxiliary_Operations

Description: The *auxiliary_operations* parameter indicates whether a specific server typically uses direct fetch and direct store or whether it uses auxiliary operations.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *server_list_entry*

Format: Enumeration

Value	**Meaning**
DIRECT_OPERATIONS	This server always works by using direct fetch and direct store. This value is only allowed for servers at end-only role DSUs.
AUXILIARY_OPERATIONS	This server always does auxiliary operations to copy server objects to and from the general server.
EITHER	This server is capable of doing either direct fetch and direct store or performing auxiliary operations to copy to and from the general server.

Before_Date

Description: The *before_date* parameter, along with the *before_time* parameter, limits the selection of distributions to be listed to those that were originated before a certain date and time.

Verbs Supplied on: List_Queue_Entries

Verbs Returned on: None

Presence Rule: Occurs when *before_time* is present.

Before_Time

Description: The *before_date* parameter, along with the *before_time* parameter, limits the selection of distributions to be listed to those that were originated before a certain date and time.

Verbs Supplied on: List_Queue_Entries

Verbs Returned on: None

Presence Rule: Occurs when *before_date* is present.

Buffer

Description: The *buffer* holds a portion of the server object to be read or written by a server.

Verbs Supplied on: Read, Write

Verbs Returned on: None

Format: Undefined byte string

Buffer_Length

Description: The *buffer_length* is the maximum amount of the server object a server can write into a *buffer*.

Verbs Supplied on: Read

Verbs Returned on: None

Format: Unsigned binary integer (1-origin); maximum: 2**64 - 2

Capacity

Description: The *capacity* service parameter indicates the capacity requirements for the distribution. The combination of this parameter and the *capacity_comparison_operator* yields the permitted levels of capacity for the distribution.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *routing_table_entry*, *service_parms*

Format: Enumeration

Value	Meaning
ZERO	No server object present
1 MB	Route through DSUs capable of handling at least 1-megabyte server objects.
4 MB	Route through DSUs capable of handling at least 4-megabyte server objects.
16 MB	Route through DSUs capable of handling at least 16-megabyte server objects.

Capacity_Comp_Op

Description: The *capacity_comparison_operator* parameter is used to allow a range of capacity service levels for a distribution. The combination of this parameter and the *capacity* parameter yields the permitted levels of capacity for the distribution.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *routing_table_entry*, *service_parms*

Format: Enumeration

Possible Values

REQUIRE_LEVEL_GE

Clean_Up_Date

Description: The *clean_up_date* is the date portion of the *sequence_number_to_clean_up*.

Verbs Supplied on: None

Verbs Returned on: Send_Distribution

Clean_Up_Seqno

Description: The *clean_up_sequence_number* is the sequence number portion of the *sequence_number_to_clean_up*.

Verbs Supplied on: None

Verbs Returned on: Send_Distribution

Format: Signed binary integer (1-origin)

Column_To_Be_Listed

Description: A *column_to_be_listed* is one of the columns of a DSU data structure that should be returned on a List_DSU_Data verb. It is used to limit the display of a data structure to only those columns that are of interest to the issuer of the verb.

Verbs Supplied on: List_DSU_Data

Verbs Returned on: None

Format: Enumeration

Value	**Data Structure Found In**
AGENT	Agent List and Directory
AUXILIARY_OPERATIONS	Server List
CAPACITY	Routing Table
CAPACITY_COMP_OP	Routing Table
DATA_STREAM_FORMAT	Connection Definitions
DAY	Connection Definitions
DEFAULT_HOP_COUNT	DSU Definition
DEN	Directory
DEST_REN	Routing Table
DEST_RGN	Routing Table
DGN	Directory
DIRECTION	Connection Definitions and MU_ID Registry
GMT_OFFSET_DIRECTION	DSU Definition
GMT_OFFSET_HOURS	DSU Definition
GMT_OFFSET_MINUTES	DSU Definition
HOLD_STATE	Next-DSU Queue Definitions
HOURS	Connection Definitions
HUNDREDTHS	Connection Definitions
LOCAL_INVOCATION_INFO	Agent List, Directory, and Server List
LOCAL_OR_GMT_FLAG	Connection Definitions
LOCAL_QUEUE_TYPE	Agent List and Directory
LOGGING	DSU Definition
LU_NAME	Connection Definitions, MU_ID Registry, Next-DSU Queue Definitions, and Routing Table
MAX_DS_RECEIVES	Connection Definitions
MAX_DS_SENDS	Connection Definitions
MINUTES	Connection Definitions
MODE_NAME	Connection Definitions, MU_ID Registry, Next-DSU Queue Definitions, and Routing Table
MONTH	Connection Definitions
MU_ID	MU_ID Registry
MU_ID_STATE	MU_ID Registry
MU_ID_INSTANCE_NUMBER	MU_ID Registry

NET_ID	Connection Definitions, MU_ID Registry, Next-DSU Queue Definitions, and Routing Table
NEXT_SEQNO	Directory
ORIGINATING_HOP_COUNT	Routing Table
PRIORITY	Routing Table
PRIORITY_COMP_OP	Routing Table
PROTECTION	Routing Table
PROTECTION_COMP_OP	Routing Table
QUEUE_ENTRY_ID	MU_ID Registry
REN	Directory, DSU Definition, and Intervention List
RESTART_CAPABLE	Server List
RGN	Directory, DSU Definition, and Intervention List
SCHEDULING_DATA	Next-DSU Queue Definitions
SECONDS	Connection Definitions
SECURITY	Routing Table
SECURITY_COMP_OP	Routing Table
SERVER	Server List
YEAR	Connection Definitions

Connection

Description: A *connection* is a set of active or potential conversations between this DSU and an adjacent DSU using a given mode name.

Verbs Supplied on: List_Connections, List_Conversations, List_Distributions_Being_Received, List_Distributions_Being_Sent, Reroute_Distribution_Copies, Reset_MU_ID_Registry, Start_Connection, Terminate_Connection

Verbs Returned on: List_Connections

Subtables Found in: *queue_ID*

Connection_Definitions_Entry

Description: The *connection_definitions_entry* describes one row in the Connection Definitions data structure.

Verbs Supplied on: Add_DSU_Data, List_DSU_Data, Modify_DSU_Data, Remove_DSU_Data

Verbs Returned on: List_DSU_Data

Connection_Queue_Type

Description: The *connection_queue_type* indicates the type of queue that is being selected.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *queue_ID*

Format: Enumeration

Possible Values

NEXT-DSU
CONTROL_MU
MID_MU_RESTART
ROUTER_DIRECTOR

Control_Info

Description: The *control_information* parameter provides the relevant information about an entry on a Control MU queue.

Verbs Supplied on: None

Verbs Returned on: List_Control_MU_Queue

Conversation_ID

Description: The *conversation_identifier* is a system-specific method for identifying a specific conversation.

Verbs Supplied on: Terminate_Conversation

Verbs Returned on: List_Conversations

Format: Undefined byte string

Conversation_Info

Description: The *conversation_information* parameter contains the relevant information for a specific conversation.

Verbs Supplied on: None

Verbs Returned on: List_Conversations

Current_Unit_Of_Work_ID

Description: The *current_unit_of_work_identifier* gives the present *unit_of_work_identifier*. If a *new_unit_of_work_identifier* is also supplied on an Assign_Read_Access verb, it replaces the *current_unit_of_work_identifier*.

Verbs Supplied on: Assign_Read_Access

Verbs Returned on: None

Data_Length

Description: The *data_length* is the length of a portion of the server object found in the *buffer*. For a server reading an object into the buffer, it can be no longer than *buffer_length*.

Verbs Supplied on: Write

Verbs Returned on: Read

Format: Unsigned binary integer (1-origin); maximum: 2**64 - 2

Data_Stream_Format

Description: The *data_stream_format* flag indicates the type of data stream that should be used on a specified connection. For more details, see Appendix D.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *connection_definitions_entry*

Format: Enumeration

Possible Values

FORMAT_SET_1
FORMAT_SET_2
ERROR

Data_Structure

Description: The *data_structure* is one of the DSU data structures that can be viewed and changed with the DS system-definition verbs.

Verbs Supplied on: Add_DSU_Data, List_DSU_Data, Modify_DSU_Data, Remove_DSU_Data

Verbs Returned on: None

Format: Enumeration

Possible Values

AGENT_LIST
CONNECTION_DEFINITIONS
DIRECTORY
DSU_DEFINITION
INTERVENTION_LIST
MU_ID_REGISTRY
NEXT-DSU_QUEUE_DEFINITIONS
ROUTING_TABLE
SERVER_LIST

Day

Description: The *day* parameter gives the day portion of the date.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *connection_definitions_entry*, *date*

Format: Signed binary integer (1-origin)

Default_Hop_Count

Description: The *default_hop_count* is the hop count used on all distributions originated at this DSU, unless overridden by an *originating_hop_count* for a specific route segment in the routing table.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *DSU_definition_entry*

Format: Signed binary integer (1-origin)

DEN

Description: The *DEN* is the second part of the user name.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *directory_entry*

Format: Character string

CGCSGIDs: 01134-00500 (Base), 00930-00500 (Enhanced Char Set)

String Conventions:

Base: Leading, imbedded, and trailing space (X′40′) characters are not allowed.

ECS: Leading space (X′40′) characters are disallowed, trailing space (X′40′) characters are not significant, and imbedded space (X′40′) characters are significant.

Dest_Agent

Description: The *destination_agent* is the transaction program at the destination DSU to which the distribution is to be delivered.

Verbs Supplied on: Send_Distribution

Verbs Returned on: Get_Distribution_Info, Receive_Distribution

Subtables Found in: *queue_ID*

Format: Character string, except for first byte

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X′40′) characters are not allowed.

The first byte of an SNA-registered transaction program name ranges in value from X′00′ to X′3F′. When the first byte ranges in value from X′41′ to X′FF′, the transaction program is not SNA registered. X′40′ is not a valid first byte.

Dest_DEN

Description: The *destination_DEN* is the second part of the name of a destination user.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *destination*, *queue_ID*

Format: Character string

CGCSGIDs: 01134-00500 (Base), 00930-00500 (Enhanced Char Set)

String Conventions:

Base Leading, imbedded, and trailing space (X'40') characters are not allowed.

ECS Leading space (X'40') characters are disallowed, trailing space (X'40') characters are not significant, and imbedded space (X'40') characters are significant.

Dest_DGN

Description: The *destination_DGN* is the first part of the name of a destination user.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *destination*, *queue_ID*

Format: Character string

CGCSGIDs: 01134-00500 (Base), 00930-00500 (Enhanced Char Set)

String Conventions:

Base Leading, imbedded, and trailing space (X'40') characters are not allowed.

ECS Leading space (X'40') characters are disallowed, trailing space (X'40') characters are not significant, and imbedded space (X'40') characters are significant.

Dest_DSU

Description: The *destination_DSU* is the name of one of the DSUs to which the distribution is to be sent.

Verbs Supplied on: List_Connections

Verbs Returned on: None

Subtables Found in: *destination*

Dest_REN

Description: The *destination_REN* is the second part of a destination DSU name. This is typically, but not necessarily, the LU name.

Verbs Supplied on: List_Connections

Verbs Returned on: None

Subtables Found in: *destination*, *routing_table_entry*

Format: Character string

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

Dest_RGN

Description: The *destination_RGN* is the first part of a destination DSU name. This is typically, but not necessarily, the network ID.

Verbs Supplied on: List_Connections

Verbs Returned on: None

Subtables Found in: *destination*, *routing_table_entry*

Format: Character string

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

Dest_User

Description: The *destination_user* is the name of one of the users to which the distribution is to be sent.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *destination*, *queue_ID*

Destination

Description:	The *destination* is one of the intended recipients of a distribution.
Verbs Supplied on:	Send_Distribution
Verbs Returned on:	Get_Distribution_Info, Get_Distribution_Log_Entry, Receive_Distribution
Note:	For Send_Distribution, either *dest_DSU* or *dest_user* is specified. For Get_Distribution_Info, Get_Distribution_Log_Entry, and Receive_Distribution, *dest_DSU* is always present and *dest_user* is optional.

DGN

Description:	The *DGN* is the first part of the user name.
Verbs Supplied on:	None
Verbs Returned on:	None
Subtables Found in:	*directory_entry*
Format:	Character string

CGCSGIDs:	01134-00500 (Base), 00930-00500 (Enhanced Char Set)	
String Conventions:		
	Base	Leading, imbedded, and trailing space (X'40') characters are not allowed.
	ECS	Leading space (X'40') characters are disallowed, trailing space (X'40') characters are not significant, and imbedded space (X'40') characters are significant.

Direction

Description:	The *direction* flag tells whether a particular conversation or connection is being used for sending or receiving.
Verbs Supplied on:	List_Control_MU_Queue
Verbs Returned on:	Get_Distribution_Log_Entry, Get_Exception_Log_Entry, List_Conversations
Subtables Found in:	*connection_definitions_entry*, *MU_ID_registry_entry*, *queue_ID*
Format:	Enumeration

Value	Meaning
SENDING	This conversation or connection is being used by DS_Send for sending MUs.
RECEIVING	This conversation or connection is being used by DS_Receive for receiving MUs.

Directory_Entry

Description:	The *directory_entry* describes one row in the Directory data structure.
Verbs Supplied on:	Add_DSU_Data, List_DSU_Data, Modify_DSU_Data, Remove_DSU_Data
Verbs Returned on:	List_DSU_Data

Distribution_ID

Description:	The *distribution_identification* is the unique identifier for a distribution. It is assigned at the origin DSU, and is used for all references to that distribution as it moves through the network. The *distribution_identification* applies to all copies of the distribution that may be generated by fan-out, and to any reports that are generated during the progress of the distribution.
Verbs Supplied on:	Assign_Read_Access, Get_Distribution_Log_Entry, Get_Exception_Log_Entry, Initiate_Read, Initiate_Write, List_Queue_Entries, List_Queues_Containing_Distribution, Query_Distribution_Sending, Receive_Distribution, Receive_Distribution_Report, Release_Read_Access, Reroute_Distribution_Copies, Send_Distribution, Sending_Sequence_Completed
Verbs Returned on:	Get_Distribution_Info, Get_Distribution_Log_Entry, Get_Exception_Log_Entry, List_Distributions_Being_Received, List_Distributions_Being_Sent, List_Queue_Entries, Receive_Distribution, Receive_Distribution_Report

Distribution_Info

Description:	The *distribution_information* parameter provides the relevant information on a distribution.
Verbs Supplied on:	None
Verbs Returned on:	List_Distributions_Being_Sent, List_Distributions_Being_Received, List_Queue_Entries

Distribution_Log_Data

Description: The *distribution_log_data* contains the data that was logged with a distribution.

Verbs Supplied on: None

Verbs Returned on: Get_Distribution_Log_Entry

Format: Product defined byte string

Distribution_Queue_ID

Description: The *distribution_queue_identifier* is the name of a local delivery queue. It is either a user name or an agent name.

Verbs Supplied on: Obtain_Local_Server_Report, Receive_Distribution, Receive_Distribution_Report, Receiving_Sequence_Completed

Verbs Returned on: None

Distribution_Time

Description: The *distribution_time* is the time at which the distribution originated.

Verbs Supplied on: None

Verbs Returned on: Get_Distribution_Info, List_Distributions_Being_Received, List_Distributions_Being_Sent, List_Queue_Entries, Receive_Distribution, Send_Distribution

DSU_Definition_Entry

Description: The *DSU_definition_entry* describes the fields in the DSU Definition data structure.

Verbs Supplied on: Add_DSU_Data, List_DSU_Data, Modify_DSU_Data, Remove_DSU_Data

Verbs Returned on: List_DSU_Data

Exception_Log_Data

Description: The *exception_log_data* contains the data that was logged for an exception.

Verbs Supplied on: None

Verbs Returned on: Get_Exception_Log_Entry

Format: Undefined byte string

Exception_Report_Req

Description: The *exception_report_requested* parameter indicates whether or not exception reports were requested on a distribution.

Verbs Supplied on: Send_Distribution

Verbs Returned on: Get_Distribution_Info, Receive_Distribution

Format: Enumeration

Value	**Meaning**
YES	Exception reporting is requested. This value is allowed only for high-integrity distributions.
NO	Exception reporting is not requested.

GMT_Offset_Direction

Description: The *GMT_offset_direction* parameter indicates whether a DSU's local time is earlier or later than GMT.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *DSU_definition_entry*

Format: Enumeration

Value	**Meaning**
EARLIER	Local time is earlier than GMT.
LATER	Local time is later than GMT.

GMT_Offset_Hours

Description: The *GMT_offset_hours* is the hours portion of the local DSU's offset from GMT.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *DSU_definition_entry*

Format: Signed binary integer (1-origin)

GMT_Offset_Minutes

Description: The *GMT_offset_minutes* is the minutes portion of the local DSU's offset from GMT.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *DSU_definition_entry*

Format: Signed binary integer (1-origin)

Hold_State

Description: The *hold_state* indicates whether a hold has been put on a next-DSU queue, and if so, whether the hold was generated in response to an exception or generated by an operator.

Verbs Supplied on: None

Verbs Returned on: List_Connections, List_Queue_Entries

Subtables Found in: *next-DSU_queue_definitions_entry*

Format: Enumeration

Value	Meaning
NOT_HELD	No hold has been placed on this queue.
EXCEPTION_HELD	A hold has been placed on this queue by the DSU as the result of an exception.
OPERATOR_HELD	A hold has been placed on this queue by the operator.

Hop_Count

Description: The *hop_count* is the remaining number of hops that may be traversed by a DS distribution on its way toward its destination DSU(s). The *hop_count* is set by the origin DSU for a distribution and by the reporting DSU for a distribution report. The *hop_count* is decremented by 1 in every DSU through which the distribution passes. If the *hop_count* reaches 0 at an intermediate DSU, exception processing is invoked.

Verbs Supplied on: None

Verbs Returned on: Get_Distribution_Info

Format: Signed binary integer (1-origin); maximum: 32,767

Hours

Description: The *hours* parameter gives the hours portion of the time.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *connection_definitions_entry*, *time*

Format: Signed binary integer (1-origin)

Hundredths

Description: The *hundredths* parameter gives the hundredths-of-a-second portion of the time.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *connection_definitions_entry*, *time*

Format: Signed binary integer (1-origin)

If_Nonunique_Key

Description: The *if_nonunique_key* flag instructs the DSU on how to proceed if a request to change a DSU data structure finds more than one row that match the key given on the request.

Verbs Supplied on: Modify_DSU_Data, Remove_DSU_Data

Verbs Returned on: None

Format: Enumeration

Value	Meaning
MODIFY_ALL	Modify all existing rows that match the key to have the new row values.
MODIFY_FIRST	Modify only the first matching row to have the new row values.
MODIFY_AND_REMOVE	Modify the first matching row to have the new row values; remove all other matching rows.
MODIFY_NONE	Do not modify any rows.
REMOVE_ALL	Remove all existing rows that match the key.
REMOVE_FIRST	Remove the first matching row.
REMOVE_NONE	Do not remove any rows.

Integrity

Description: The *integrity* flag indicates whether DS should use confirmation protocols on the conversation and at the PB to prevent losses or duplications of this distribution. For more details, see Chapter 2.

Verbs Supplied on: Send_Distribution

Verbs Returned on: Get_Distribution_Info, Receive_Distribution, Receive_Distribution_Report

Format: Enumeration

Value	Meaning
HIGH	The sending agent uses Sending_Sequence_Completed to manage the transfer of the distribution from the agent to the DSU. Also, the DSU uses confirmation flows and MU_IDs when sending the distribution from one hop to the next.
BASIC	All responsibility for preventing losses and duplicates of the distribution rests with the agent. The distribution service makes no extra effort to guard against losses and duplicates during transmission of the distribution. Exception reporting can not be requested for a basic-integrity distribution.

Intervention_List_Entry

Description: The *intervention_list_entry* describes one row in the Intervention List data structure.

Verbs Supplied on: Add_DSU_Data, List_DSU_Data, Modify_DSU_Data, Remove_DSU_Data

Verbs Returned on: List_DSU_Data

Last_Byte_Received

Description: The *last_byte_received* is the last byte received by the receiving DSU before the MU was suspended. The byte count begins with the first byte of atomic data within the encompassing structure. A byte count of X'FFFFFFFFFFFFFFFF' indicates that the structure was fully received. The byte count contains only atomic data and does not contain the segmenting LLs for segmented structures.

Verbs Supplied on: None

Verbs Returned on: Query_Last_Byte_Received

Format: Unsigned binary integer (1-origin)

Local_Info

Description: The *local_info* is the information that is kept for a local program, such as an agent or server. This could be a module name, default parameters, or any other required system-specific information.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *agent_list_entry*, *directory_entry*, *server_list_entry*

Format: Undefined byte string

Local_Or_GMT_Flag

Description: The *local_or_GMT_flag* tells how a time value that passes across the protocol boundary should be interpreted.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *connection_definitions_entry*, *time*

Format: Enumeration

Value	Meaning
LOCAL	The time is the local time value for this DSU.
GMT	The time is a GMT time value.
ORIGIN_LOCAL	The time is the local time value at the origin DSU of the distribution.

Local_Queue_Type

Description: The *local_queue_type* is the type of queue that is used to deliver distributions to a specific user by a specific agent.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *agent_list_entry*, *directory_entry*

Format: Enumeration

Value	Meaning
AGENT	The queue is identified by the agent name.
USER	The queue is identified by the user name.

Logging

Description: The *logging* flag specifies whether or not the DSU is currently doing optional distribution logging.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *DSU_definition_entry*

Format: Enumeration

Value	Meaning
YES	Perform optional distribution logging.
NO	Do not perform optional distribution logging.

Logging_Date

Description: The *logging_date*, along with the *logging_time*, specifies the date and time a log entry was generated.

Verbs Supplied on: None

Verbs Returned on: Get_Distribution_Log_Entry, Get_Exception_Log_Entry

Logging_Time

Description: The *logging_time*, along with the *logging_date*, specifies the date and time a log entry was generated.

Verbs Supplied on: None

Verbs Returned on: Get_Distribution_Log_Entry, Get_Exception_Log_Entry

LU_Name

Description: The *logical_unit_name* is the second part of a network-qualified LU-name. It is usually, though not always, the same as the REN.

Verbs Supplied on: Initiate_Read, Initiate_Write, List_Connections, List_Control_MU_Queue, List_Conversations, List_Distributions_Being_Received, List_Distributions_Being_Sent, Query_Last_Byte_Received, Reroute_Distribution_Copies, Reset_MU_ID_Registry, Start_Connection, Terminate_Connection, Terminate_Restartability

Verbs Returned on: Get_Distribution_Log_Entry, Get_Exception_Log_Entry, List_Connections, List_Conversations

Subtables Found in: *connection_definitions_entry*, *MU_ID_registry_entry*, *next-DSU_queue_definitions_entry*, *queue_ID*, *routing_table_entry*

Format: Character string

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

Max_DS_Receives

Description: The *maximum_DS_Receives* is the largest number of instances of DS_Receive that can be active on a connection.

Verbs Supplied on: Start_Connection

Verbs Returned on: None

Subtables Found in: *connection_definitions_entry*

Presence Rule: Occurs when the value of *direction* is RECEIVING.

Format: Signed binary integer (1-origin)

Max_DS_Sends

Description: The *maximum_DS_Sends* is the largest number of instances of DS_Send that can be active on a connection. It should typically be less than or equal to the number of *conwinner* sessions the DSU has for that connection.

Verbs Supplied on: Start_Connection

Verbs Returned on: None

Subtables Found in: *connection_definitions_entry*

Presence Rule: Occurs when the value of *direction* is SENDING.

Format: Signed binary integer (1-origin)

Minutes

Description: The *minutes* parameter gives the minutes portion of the time.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *connection_definitions_entry*, *time*

Format: Signed binary integer (1-origin)

Mode_Name

Description: The *mode_name* identifies the type of traffic that is appropriate for a specific conversation.

Verbs Supplied on: Initiate_Read, Initiate_Write, List_Connections, List_Control_MU_Queue, List_Conversations, List_Distributions_Being_Received, List_Distributions_Being_Sent, Query_Last_Byte_Received, Reroute_Distribution_Copies, Reset_MU_ID_Registry, Start_Connection, Terminate_Connection, Terminate_Restartability

Verbs Returned on: Get_Distribution_Log_Entry, Get_Exception_Log_Entry, List_Connections, List_Conversations

Subtables Found in: *connection_definitions_entry*, *MU_ID_registry_entry*, *next-DSU_queue_definitions_entry*, *queue_ID*, *routing_table_entry*

Format: Undefined byte string

Modified_Row

Description: A *modified_row* replaces one or more rows specified by *row_selection_criteria* on a Modify_DSU_Data verb.

Verbs Supplied on: Modify_DSU_Data

Verbs Returned on: None

Month

Description: The *month* parameter gives the month portion of the date.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *connection_definitions_entry*, *date*

Format: Signed binary integer (1-origin)

MU_Action

Description: The *message_unit_action* flag indicates what action should be taken on distributions that are currently being sent or received on a conversation (or connection) that is being terminated.

Verbs Supplied on: Terminate_Connection, Terminate_Conversation

Verbs Returned on: None

Format: Enumeration

Value	**Meaning**
ABORT	Abort all distributions being sent or received on the conversation or connection.
COMPLETE	Complete any distributions currently being sent or received on the conversation or connection, but do not begin any new distributions.
SUSPEND	Suspend any distributions currently being sent or received on the conversation or connection for later restart.

MU_ID

Description: The *message_unit_identifier* is a number that uniquely identifies a distribution MU throughout its existence. An MU exists for only one hop, from one DSU to the adjacent DSU. An MU_ID is unique only for a particular *LU name*, *mode name* combination.

Verbs Supplied on: Initiate_Read, Initiate_Write, Query_Last_Byte_Received, Terminate_Restartability

Verbs Returned on: Get_Distribution_Info, Get_Distribution_Log_Entry, Get_Exception_Log_Entry, List_Control_MU_Queue

Subtables Found in: *MU_ID_registry_entry*

Format: Signed binary integer (1-origin)

MU_ID_Registry_Entry

Description: The *MU_ID_registry_entry* describes one row in the MU_ID Registry data structure.

Verbs Supplied on: Add_DSU_Data, List_DSU_Data, Modify_DSU_Data, Remove_DSU_Data

Verbs Returned on: List_DSU_Data

MU_ID_State

Description: The *message_unit_identifier_state* indicates the state of processing, on either the send or receive side, for an MU_ID. For details of the meaning of each state, see Chapter 2.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *MU_ID_registry_entry*

Format: Enumeration

Value	Send or Receive Side Occurrence
COMPLETED	Receive side
CQMU_PENDING	Send side
IN_TRANSIT	Both send and receive sides
NOT_ASSIGNED	Send side
NOT_RECEIVED	Receive side
PURGED	Both send and receive sides

RETRY_PENDING	Send side
SUSPENDED	Both send and receive sides
TERMINATED	Receive side
TERMINATION_PENDING	Send side
TRANSFER_PENDING	Send side

MU_Instance_Number

Description: The *message_unit_instance_number* identifies the instance of a particular distribution message unit and its corresponding MU_ID.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *MU_ID_registry_entry*

Format: Signed binary integer (1-origin)

MU_Type

Description: The *MU_type* indicates the type of MU that is found on a Control MU queue.

Verbs Supplied on: None

Verbs Returned on: List_Control_MU_Queue

Format: Enumeration

Value	**Meaning**
SEMU	Sender-Exception Message Unit
REMU	Receiver-Exception Message Unit
CQMU	Completion-Query Message Unit
CRMU	Completion-Report Message Unit
PRMU	Purge-Report Message Unit
RRMU	Reset-Request Message Unit
RAMU	Reset-Accepted Message Unit

Net_ID

Description: The *network_identifier* is the first part of a network-qualified LU-name. It is usually, though not always, the same as the RGN.

Verbs Supplied on: Initiate_Read, Initiate_Write, List_Connections, List_Control_MU_Queue, List_Conversations, List_Distributions_Being_Received, List_Distributions_Being_Sent, Query_Last_Byte_Received, Reroute_Distribution_Copies, Reset_MU_ID_Registry, Start_Connection, Terminate_Connection, Terminate_Restartability

Verbs Returned on: Get_Distribution_Log_Entry, Get_Exception_Log_Entry, List_Connections, List_Conversations

Subtables Found in: *connection_definitions_entry*, *MU_ID_registry_entry*, *next-DSU_queue_definitions_entry*, *queue_ID*, *routing_table_entry*

Format: Character string

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

New_Row

Description: A *new_row* is a row being added to a DSU data structure by the Add_DSU_Data verb.

Verbs Supplied on: Add_DSU_Data

Verbs Returned on: None

New_Row_Number

Description: The *new_row_number* is the next row that matches the *row_selection_criteria* specified on a List_DSU_Data verb. It is used when there are too many matching rows to return on one invocation of the List_DSU_Data verb. To obtain further information, the agent reissues List_DSU_Data, using the value returned in *new_row_number* as the *starting_row_number*.

Verbs Supplied on: None

Verbs Returned on: List_DSU_Data

Format: Signed binary integer (1-origin)

New_Unit_Of_Work_ID

Description: The *new_unit_of_work_identifier* provides the value that should replace the *current_unit_of_work_identifier*.

Verbs Supplied on: Assign_Read_Access

Verbs Returned on: None

Next-DSU_Queue_Definitions_Entry

Description: The *next-DSU_queue_definitions_entry* describes one row in the Next-DSU Queue Definitions data structure.

Verbs Supplied on: Add_DSU_Data, List_DSU_Data, Modify_DSU_Data, Remove_DSU_Data

Verbs Returned on: List_DSU_Data

Next_MU_ID

Description: The *next_message_unit_identifier* indicates which MU_ID should be expected next when the two MU_ID registries for a connection are synchronized.

Verbs Supplied on: Reset_MU_ID_Registry

Verbs Returned on: None

Format: Signed binary integer (1-origin)

Next_Queue_Entry_ID

Description: The *next_queue_entry_identifier* indicates the next queue_entry_identifier to be listed on a List_Queue_Entries verb. It occurs when there were too many distributions to be listed on one invocation of the List_Queue_Entries verb.

Verbs Supplied on: None

Verbs Returned on: List_Control_MU_Queue, List_Queue_Entries

Format: Undefined byte string

Next_Seqno

Description: The *next_sequence_number* is the sequence number to be used on the next distribution sent on this date for a specific user by a specific agent. The sequence number is reset to 1 for the first distribution each day.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *directory_entry*

Format: Signed binary integer (1-origin)

Number_Of_Matching_Entries

Description: The *number_of_matching_entries* indicates how many log entries match the given distribution identification.

Verbs Supplied on: None

Verbs Returned on: Get_Distribution_Log_Entry, Get_Exception_Log_Entry

Format: Signed binary integer (1-origin)

Origin_Agent

Description: The *origin_agent* is the transaction program at the origin DSU that originated the distribution.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *distribution_ID*

Format: Character string, except for first byte

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X′40′) characters are not allowed.

The first byte of an SNA-registered transaction program name ranges in value from X′00′ to X′3F′. When the first byte ranges in value from X′41′ to X′FF′, the transaction program is not SNA registered. X′40′ is not a valid first byte.

Origin_Date

Description: The *origin_date* is the date on which the distribution originated.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *distribution_ID*

Origin_DEN

Description: The *origin_DEN* is the second part of the user name of the distribution originator.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *distribution_ID*

Format: Character string

CGCSGIDs: 01134-00500 (Base), 00930-00500 (Enhanced Char Set)

String Conventions:

Base Leading, imbedded, and trailing space (X'40') characters are not allowed.

ECS Leading space (X'40') characters are disallowed, trailing space (X'40') characters are not significant, and imbedded space (X'40') characters are significant.

Origin_DGN

Description: The *origin_DGN* is the first part of the user name of the distribution originator.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *distribution_ID*

Format: Character string

CGCSGIDs: 01134-00500 (Base), 00930-00500 (Enhanced Char Set)

String Conventions:

Base — Leading, imbedded, and trailing space (X'40') characters are not allowed.

ECS — Leading space (X'40') characters are disallowed, trailing space (X'40') characters are not significant, and imbedded space (X'40') characters are significant.

Origin_DSU

Description: The *origin_DSU* is the name of the DSU at which the distribution originated.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *distribution_ID*

Origin_REN

Description: The *origin_REN* is the second part of the name of the DSU at which the distribution originated. This is typically, but not necessarily, the LU name.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *distribution_ID*

Format: Character string

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

Origin_RGN

Description: The *origin_RGN* is the first part of the name of the DSU at which the distribution originated. This is typically, but not necessarily, the network ID.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *distribution_ID*

Format: Character string

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

Origin_Seqno

Description: The *origin_sequence_number* is the number assigned to the distribution by the origin agent as part of the *distribution_identification*. For FS2, the number ranges from 1 to (2**31)-1. For FS1, the number ranges from 0 (for report MUs) to 9999. Refer to Appendix D for migration details.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *distribution_ID*

Format: Signed binary integer (1-origin)

Origin_User

Description: The *origin_user* is the user name of the originator of the distribution.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *distribution_ID*

Originating_Hop_Count

Description: The *originating_hop_count* specifies the hop count to be used for distributions on a particular route segment. It overrides the *default_hop_count* for the DSU.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *routing_table_entry*

Format: Signed binary integer (1-origin)

Parent_Class

Description:	The *parent_class* is the class of a parent structure. For more details on the classes of structures, see Appendix G.
Verbs Supplied on:	None
Verbs Returned on:	None
Subtables Found in:	*SNA_condition_report*
Format:	Enumeration

Possible Values

LENGTH_BOUNDED_LLID
LENGTH_BOUNDED_LT
DELIMITED_LLID
DELIMITED_LT
IMPLIED_LLID
IMPLIED_LT

Parent_ID_Or_T

Description:	The *parent_ID_or_T* is the ID or T value of a parent structure. ID's are the registered GDS codepoints and are referred to in SNA Formats. T's are architecture-specific values relative to the encompassing ID.
Verbs Supplied on:	None
Verbs Returned on:	None
Subtables Found in:	*SNA_condition_report*
Format:	Undefined byte string

Parent_Instance

Description:	The *parent_instance* is used when a parent structure occurs multiple times. The value of *parent_instance* identifies the particular instance within a position.
Verbs Supplied on:	None
Verbs Returned on:	None
Subtables Found in:	*SNA_condition_report*
Format:	Signed binary integer (1-origin)

Parent_Position

Description: The *parent_position* is the position of this parent structure within its parent (if one exists) in this particular MU. Multiple consecutive instances of a repeatable parent structure share a single position; they can be distinguished by *parent_instance*.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *SNA_condition_report*

Format: Signed binary integer (1-origin)

Parent_Spec

Description: The *parent_specification* contains the identifier (ID or T) and the class of a parent structure. For a parent structure that occurs multiple times, the instance may also be included. The value of the *parent_instance* identifies the particular instance. The position of this parent structure within its parent (if one exists) may also be included. This would typically be done when this parent structure is an unordered child of its parent.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *SNA_condition_report*

Previously_Received_Date

Description: The *previously_received_date* is the date portion of the *previously_received_date_time*.

Verbs Supplied on: None

Verbs Returned on: Get_Distribution_Info, List_Queue_Entries, Receive_Distribution, Receive_Distribution_Report

Previously_Received_Date_Time

Description: The *previously_received_date_time* indicates when this distribution or report had been earlier received. The previous receive verb was never acknowledged with a Receiving_Sequence_Completed verb, and the DSU is now trying to account for this failure to acknowledge receipt of the distribution or report.

Verbs Supplied on: None

Verbs Returned on: Get_Distribution_Info, List_Queue_Entries, Receive_Distribution, Receive_Distribution_Report

Previously_Received_Time

Description: The *previously_received_time* is the time portion of the *previously_received_date_time.*

Verbs Supplied on: None

Verbs Returned on: Get_Distribution_Info, List_Queue_Entries, Receive_Distribution, Receive_Distribution_Report

Priority

Description: The *priority* service parameter indicates the priority requirements for the distribution. The combination of this parameter and the *priority_comparison_operator* yields the permitted levels of priority for the distribution.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *report_service_parms*, *routing_table_entry*, *service_parms*

Format: Enumeration

Note: For the Folding Data Priorities elective, DATAHI replaces DATA_9 through DATA_16 and DATALO replaces DATA_1 through DATA_8.

Possible Values

FAST
CONTROL
DATA_16
DATA_15
DATA_14
DATA_13
DATA_12
DATA_11
DATA_10
DATA_9
DATA_8
DATA_7
DATA_6
DATA_5
DATA_4
DATA_3
DATA_2
DATA_1

Priority_Comp_Op

Description:	The *priority_comparison_operator* parameter is used to allow a range of priority service levels for a distribution. The combination of this parameter and the *priority* parameter yields the permitted levels of priority for the distribution.
Verbs Supplied on:	None
Verbs Returned on:	None
Subtables Found in:	*report_service_parms*, *routing_table_entry*, *service_parms*
Format:	Enumeration

Possible Values

REQUIRE_LEVEL_GE

Product_Specific_Data

Description:	The *product_specific_data* specifies product data that is associated with the distribution, or which could be helpful in problem determination.
Verbs Supplied on:	None
Verbs Returned on:	Get_Distribution_Log_Entry, Get_Exception_Log_Entry
Format:	Undefined byte string

Program_Name

Description: The *program_name* specifies the name of the program that generated the log entry.

Verbs Supplied on: None

Verbs Returned on: Get_Distribution_Log_Entry, Get_Exception_Log_Entry

Format: Character string, except for first byte

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X′40′) characters are not allowed.

The first byte of an SNA-registered transaction program name ranges in value from X′00′ to X′3F′. When the first byte ranges in value from X′41′ to X′FF′, the transaction program is not SNA registered. X′40′ is not a valid first byte.

Protection

Description: The *protection* service parameter indicates the protection requirements for the distribution. The combination of this parameter and the *protection_comparison_operator* yields the permitted levels of protection for the distribution.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *report_service_parms*, *routing_table_entry*, *service_parms*

Format: Enumeration

Value	**Meaning**
LEVEL1	Safe store is not performed.
LEVEL2	Safe store must be performed.

Protection_Comp_Op

Description: The *protection_comparison_operator* parameter is used to allow a range of protection service levels for a distribution. The combination of this parameter and the *protection* parameter yields the permitted levels of protection for the distribution.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *report_service_parms*, *routing_table_entry*, *service_parms*

Format: Enumeration

Possible Values

REQUIRE_LEVEL_GE

Querying_Agent

Description:	The *querying_agent* is the agent that issues a List_Queue_Entries verb. It might not be the same agent that originated the distribution.
Verbs Supplied on:	List_Queue_Entries
Verbs Returned on:	None
Format:	Character string, except for first byte

CGCSGID:	01134-00500 (Character Set AR)
String Conventions:	Leading, imbedded, and trailing space (X'40') characters are not allowed. The first byte of an SNA-registered transaction program name ranges in value from X'00' to X'3F'. When the first byte ranges in value from X'41' to X'FF', the transaction program is not SNA registered. X'40' is not a valid first byte.

Queue_Entry_ID

Description:	The *queue_entry_identifier* specifies one particular entry in a given queue. The combination of *queue_identifier* and *queue_entry_identifier* uniquely identify any distribution copy in the DSU.
Verbs Supplied on:	Get_Distribution_Info, Hold_Distribution_Copy, Obtain_Local_Server_Report, Purge_Queue_Entry, Receive_Distribution, Receive_Distribution_Report, Receiving_Sequence_Completed, Release_Distribution_Copy
Verbs Returned on:	List_Control_MU_Queue, List_Queue_Entries, Obtain_Local_Server_Report, Receive_Distribution, Receive_Distribution_Report
Subtables Found in:	*MU_ID_registry_entry*
Format:	Undefined byte string

Queue_Entry_Type

Description:	The *queue_entry_type* specifies what is found in a given queue entry.
Verbs Supplied on:	List_Queue_Entries
Verbs Returned on:	List_Queue_Entries
Format:	Enumeration

Possible Values

DISTRIBUTION
DISTRIBUTION_REPORT
SERVER_REPORT

Queue_ID

Description: The *queue_identifier* specifies one particular local delivery queue or next-DSU queue. The combination of *queue_identifier* and *queue_entry_identifier* uniquely identify any distribution copy in the DSU.

Verbs Supplied on: Get_Distribution_Info, Hold_Distribution_Copy, List_Queue_Entries, Purge_Queue_Entry, Release_Distribution_Copy

Verbs Returned on: List_Queues_Containing_Distribution

Received_Server_Bytes

Description: The *received_server_bytes* parameter indicates how many bytes of the server object have been received. This parameter is applicable only to a distribution currently being received at this DSU.

Verbs Supplied on: None

Verbs Returned on: List_Distributions_Being_Received

Format: Unsigned binary integer (1-origin); maximum: 2**64 - 2

Receiving_Agent

Description: The *receiving_agent* is the agent that receives a distribution, distribution report, or server report.

Verbs Supplied on: Obtain_Local_Server_Report, Receive_Distribution, Receive_Distribution_Report

Verbs Returned on: None

Format: Character string, except for first byte

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X′40′) characters are not allowed.

The first byte of an SNA-registered transaction program name ranges in value from X′00′ to X′3F′. When the first byte ranges in value from X′41′ to X′FF′, the transaction program is not SNA registered. X′40′ is not a valid first byte.

Receiving_DSU

Description: The *receiving_DSU* is the name of the DSU that was receiving a distribution about which a report is being generated.

Verbs Supplied on: None

Verbs Returned on: Receive_Distribution_Report

Receiving_REN

Description: The *receiving_REN* is the second part of the name of the DSU that was receiving a distribution about which a report is being generated. This is typically, but not necessarily, the LU name.

Verbs Supplied on: None

Verbs Returned on: Receive_Distribution_Report

Format: Character string

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

Receiving_RGN

Description: The *receiving_RGN* is the first part of the name of the DSU that was receiving a distribution about which a report is being generated. This is typically, but not necessarily, the network ID.

Verbs Supplied on: None

Verbs Returned on: Receive_Distribution_Report

Format: Character string

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

Remaining_Server_Bytes

Description: The *remaining_server_bytes* parameter indicates how many bytes of the server object remain to be sent. This parameter is applicable only to a distribution currently being sent by this DSU.

Verbs Supplied on: None

Verbs Returned on: List_Distributions_Being_Sent

Format: Unsigned binary integer (1-origin); maximum: 2**64 - 2

REN

Description: The *REN* is the second part of the DSU name. This is typically, but not necessarily, the LU name.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *directory_entry*, *DSU_definition_entry*, *intervention_list_entry*

Format: Character string

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X′40′) characters are not allowed.

Report-To_Agent

Description: The *report-to_agent* is the name of the application transaction program to be started after the report is queued for delivery.

Verbs Supplied on: Send_Distribution

Verbs Returned on: Get_Distribution_Info, Receive_Distribution, Receive_Distribution_Report

Format: Character string, except for first byte

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X′40′) characters are not allowed.

The first byte of an SNA-registered transaction program name ranges in value from X′00′ to X′3F′. When the first byte ranges in value from X′41′ to X′FF′, the transaction program is not SNA registered. X′40′ is not a valid first byte.

Report-To_DEN

Description: The *report_to_DEN* is the second part of the user name to which distribution reports are to be sent.

Verbs Supplied on: Send_Distribution

Verbs Returned on: Get_Distribution_Info, Receive_Distribution, Receive_Distribution_Report

Format: Character string

CGCSGIDs: 01134-00500 (Base), 00930-00500 (Enhanced Char Set)

String Conventions:

Base Leading, imbedded, and trailing space (X'40') characters are not allowed.

ECS Leading space (X'40') characters are disallowed, trailing space (X'40') characters are not significant, and imbedded space (X'40') characters are significant.

Report-To_DGN

Description: The *report_to_DGN* is the first part of the user name to which distribution reports are to be sent.

Verbs Supplied on: Send_Distribution

Verbs Returned on: Get_Distribution_Info, Receive_Distribution, Receive_Distribution_Report

Format: Character string

CGCSGIDs: 01134-00500 (Base), 00930-00500 (Enhanced Char Set)

String Conventions:

Base Leading, imbedded, and trailing space (X'40') characters are not allowed.

ECS Leading space (X'40') characters are disallowed, trailing space (X'40') characters are not significant, and imbedded space (X'40') characters are significant.

Report-To_DSU

Description: The *report-to_DSU* is the name of the DSU to which distribution reports are to be sent.

Verbs Supplied on: Send_Distribution

Verbs Returned on: Get_Distribution_Info, Receive_Distribution, Receive_Distribution_Report

Report-To_REN

Description: The *report-to_REN* is the second part of the DSU name to which distribution reports are to be sent. This is typically, but not necessarily, the LU name.

Verbs Supplied on: Send_Distribution

Verbs Returned on: Get_Distribution_Info, Receive_Distribution, Receive_Distribution_Report

Format: Character string

CGCSGID:	01134-00500 (Character Set AR)
String Conventions:	Leading, imbedded, and trailing space (X'40') characters are not allowed.

Report-To_RGN

Description:	The *report-to_RGN* is the first part of the DSU name to which distribution reports are to be sent. This is typically, but not necessarily, the network ID.
Verbs Supplied on:	Send_Distribution
Verbs Returned on:	Get_Distribution_Info, Receive_Distribution, Receive_Distribution_Report
Format:	Character string

CGCSGID:	01134-00500 (Character Set AR)
String Conventions:	Leading, imbedded, and trailing space (X'40') characters are not allowed.

Report-To_User

Description:	The *report-to_user* is the name of the user to which distribution reports are to be sent.
Verbs Supplied on:	Send_Distribution
Verbs Returned on:	Get_Distribution_Info, Receive_Distribution, Receive_Distribution_Report

Report_Date

Description:	The *report_date* contains the date on which the *reporting_DSU* generated the report.
Verbs Supplied on:	None
Verbs Returned on:	Receive_Distribution_Report

Report_Service_Parms

Description: The *report_service_parameters* describe the service requested for the distribution report by the origin agent when the agent wants to override the service parameters that would be routinely generated by the reporting DSU for the report MU. If report service parameters are specified, they are used as the service parameters in any DRMUs that are generated as part of the distribution. If the origin agent does not provide report service parameters, a DSU that generates a report derives service parameters for the DRMU from the service parameters in the DTMU.

Verbs Supplied on: Send_Distribution

Verbs Returned on: Get_Distribution_Info, Receive_Distribution

Report_Time

Description: The *report_time* contains the time at which the *reporting_DSU* generated the report.

Verbs Supplied on: None

Verbs Returned on: Receive_Distribution_Report

Reported-On_Dest_Agent

Description: The *reported-on_destination_agent* is the name of the intended destination agent of the distribution that is being reported on.

Verbs Supplied on: None

Verbs Returned on: Receive_Distribution_Report

Presence Rule: Occurs when *dest_agent* was specified in the reported-on distribution.

Format: Character string, except for first byte

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

The first byte of an SNA-registered transaction program name ranges in value from X'00' to X'3F'. When the first byte ranges in value from X'41' to X'FF', the transaction program is not SNA registered. X'40' is not a valid first byte.

Reported-On_Dest_DEN

Description: The *reported-on_destination_DEN* is the second part of the name of one of the original destination users being reported on.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *reported_on_destination*

Format: Character string

CGCSGIDs: 01134-00500 (Base), 00930-00500 (Enhanced Char Set)

String Conventions:

Base Leading, imbedded, and trailing space (X'40') characters are not allowed.

ECS Leading space (X'40') characters are disallowed, trailing space (X'40') characters are not significant, and imbedded space (X'40') characters are significant.

Reported-On_Dest_DGN

Description: The *reported-on_destination_DGN* is the first part of the name of one of the original destination users being reported on.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *reported_on_destination*

Format: Character string

CGCSGIDs: 01134-00500 (Base), 00930-00500 (Enhanced Char Set)

String Conventions:

Base Leading, imbedded, and trailing space (X'40') characters are not allowed.

ECS Leading space (X'40') characters are disallowed, trailing space (X'40') characters are not significant, and imbedded space (X'40') characters are significant.

Reported-On_Dest_DSU

Description: The *reported-on_destination_DSU* is one of the original destination DSUs being reported on.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *reported_on_destination*

Presence Rule: Always present, unless the report passed through an FS1 subnet.

Reported-On_Dest_REN

Description: The *reported-on_destination_REN* is the second part of the name of one of the original destination DSUs being reported on. This is typically, but not necessarily, the LU name.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *reported_on_destination*

Format: Character string

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X′40′) characters are not allowed.

Reported-On_Dest_RGN

Description: The *reported-on_destination_RGN* is the first part of the name of one of the original destination DSUs being reported on. This is typically, but not necessarily, the network ID.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *reported_on_destination*

Format: Character string

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X′40′) characters are not allowed.

Reported-On_Dest_User

Description: The *reported-on_destination_user* is the name of one of the original destination users being reported on.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *reported_on_destination*

Reported-On_Destination

Description: The *reported-on_destination* is one of the distribution destinations that is being reported on.

Verbs Supplied on: None

Verbs Returned on: Get_Exception_Log_Entry

Subtables Found in: *SNA_condition_report*

Reported-On_Time

Description: The *reported-on_time* is the *distribution_time* that was returned on Send_Distribution.

Verbs Supplied on: None

Verbs Returned on: Receive_Distribution_Report

Reporting_DSU

Description: The *reporting_DSU* is the name of the DSU that generated the report.

Verbs Supplied on: None

Verbs Returned on: Receive_Distribution_Report

Reporting_REN

Description: The *reporting_REN* is the second part of the name of the DSU that generated the report. This is typically, but not necessarily, the LU name.

Verbs Supplied on: None

Verbs Returned on: Receive_Distribution_Report

Format: Character string

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X′40′) characters are not allowed.

Reporting_RGN

Description: The *reporting_RGN* is the first part of the name of the DSU that generated the report. This is typically, but not necessarily, the network ID.

Verbs Supplied on: None

Verbs Returned on: Receive_Distribution_Report

Format: Character string

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X′40′) characters are not allowed.

Requested_Entry_Number

Description: The *requested_entry_number* indicates which log entry to return when there are more than one with the same distribution identification. For example, a *requested_entry_number* of 4 would retrieve the fourth log entry for a specified distribution identification.

Verbs Supplied on: Get_Distribution_Log_Entry, Get_Exception_Log_Entry

Verbs Returned on: None

Format: Signed binary integer (1-origin)

Requesting_Process

Description: The *requesting_process* is the process that is requesting a server action.

Verbs Supplied on: Backout_Server_Object, Initiate_Read, Initiate_Write, Query_Last_Byte_Received, Terminate_Restartability

Verbs Returned on: None

Format: Character string, except for first byte

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

The first byte of an SNA-registered transaction program name ranges in value from X'00' to X'3F'. When the first byte ranges in value from X'41' to X'FF', the transaction program is not SNA registered. X'40' is not a valid first byte.

Restart_Byte

Description: The *restart_byte* indicates where the sender is beginning retransmission of the server object.

Verbs Supplied on: Initiate_Read, Initiate_Write

Verbs Returned on: None

Format: Unsigned binary integer (1-origin); maximum: 2**64 - 2

Restart_Capable

Description: The *restart_capable* flag indicates whether a server is capable of Byte-Count Restart.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *server_list_entry*

Format: Enumeration

Value	**Meaning**
YES	Server is capable of Byte-Count Restart.
NO	Server is not capable of Byte-Count Restart.

Restart_ID

Description: The *restart_ID* identifies a server object being read or written for restart purposes.

Verbs Supplied on: Initiate_Read, Initiate_Write, Query_Last_Byte_Received, Terminate_Restartability

Verbs Returned on: None

Restartability

Description: The *restartability* parameter indicates whether a server should keep checkpoints on an object that it is reading or writing, in preparation for a later restart.

Verbs Supplied on: Initiate_Read, Initiate_Write

Verbs Returned on: None

Format: Enumeration

Value	Meaning
YES	If possible, keep checkpoints on this object beirg read/written to allow mid-server object restart.
NO	Do not keep checkpoints on this object.

Return_Code

Description: The *return_code* indicates the result of a verb that was issued by an agent or by DS.

Verbs Supplied on: None

Verbs Returned on: All

Format: Enumeration

Distribution Return Codes

OK
PARAMETER_CHECK
INVALID_ORIGIN_USER_NAME
INVALID_REPORT-TO_DEST
UNSUPPORTED_SERVICE_LEVEL
INVALID_SERVER_NAME
INVALID_SERVER_PARAMETER
INVALID_QUEUE_IDENTIFIER
I/O_EXCEPTION
QUEUE_EMPTY
TEMPORARY_SERVER_EXCEPTION
SPECIFIC_SERVER_EXCEPTION
AGENT_NOT_DEFINED
TEMPORARY_EXCEPTION
TOO_MANY_OUTSTANDING_SEQNOS

Operations Return Codes

OK
PARAMETER_CHECK
I/O_EXCEPTION
TEMPORARY_EXCEPTION
MORE_DATA_TO_COME
QUEUE_ENTRY_NOT_FOUND
QUEUE_ENTRY_IN_USE
QUEUE_EMPTY
DUPLICATE_QUEUE_ENTRY_CREATED
CONNECTION_NOT_FOUND
INVALID_DATA_STRUCTURE_NAME
INVALID_KEY
INVALID_STARTING_POSITION
DATA_ENTRY_NOT_FOUND
DATA_ENTRY_IN_USE
DATA_STRUCTURE_EMPTY
DATA_STRUCTURE_FULL
DATA_ENTRY_ALREADY_EXISTS

Server Return Codes

OK
PARAMETER_CHECK
END_OF_DATA (EOD)
SPECIFIC_SERVER_EXCEPTION
I/O_EXCEPTION
TEMPORARY_SERVER_EXCEPTION
INVALID_SERVER_NAME
ACCESS_LIST_ENTRY_NOT_FOUND
NOT_RESTART_CAPABLE
RESTART_ID_NOT_FOUND
CANNOT_RESTART_AT_BYTE_POSITION
OBJECT_NOT_FOUND

Note: The PARAMETER_CHECK, I/O_EXCEPTION, TEMPORARY_EXCEPTION, and OBJECT_NOT_FOUND return codes are issued only by the general server, while SPECIFIC_SERVER_EXCEPTION is issued only by a specific server.

RGN

Description:	The *RGN* is the first part of the DSU name. This is typically, but not necessarily, the network ID.
Verbs Supplied on:	None
Verbs Returned on:	None
Subtables Found in:	*directory_entry*, *DSU_definition_entry*, *intervention_list_entry*
Format:	Character string

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

Route

Description: The *route* parameter is used to choose among the various route segments from this DSU. It is composed of a *destination_DSU* and the *service_parms* to be used to route to that destination DSU.

Verbs Supplied on: List_Connections

Verbs Returned on: None

Routing_Table_Entry

Description: The *routing_table_entry* describes one row in the Routing Table data structure.

Verbs Supplied on: Add_DSU_Data, List_DSU_Data, Modify_DSU_Data, Remove_DSU_Data

Verbs Returned on: List_DSU_Data

Row_Selection_Criteria

Description: The *row_selection_criteria* are used to allow the DS system definition verbs to operate on only a portion of a DSU data structure. The *row_selection_criteria* takes the form of a row of the table with some or all of its column values specified. The DSU restricts its operations on the data structure to those rows that match the *row_selection_criteria*.

Verbs Supplied on: List_DSU_Data, Modify_DSU_Data, Remove_DSU_Data

Verbs Returned on: None

Scheduling_Data

Description: The *scheduling_data* provides the data necessary to do time-of-day scheduling for a next-DSU queue.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *next-DSU_queue_definitions_entry*

Format: Undefined byte string

Seconds

Description: The *seconds* parameter gives the seconds portion of the time.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *connection_definitions_entry*, *time*

Format: Signed binary integer (1-origin)

Security

Description: The *security* service parameter indicates the security requirements for the distribution. The combination of this parameter and the *security_comparison_operator* yields the permitted levels of security for the distribution.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *report_service_parms*, *routing_table_entry*, *service_parms*

Format: Enumeration

Value	Meaning
LEVEL1	Security is not required.
LEVEL2	Security is required.

Security_Comp_Op

Description: The *security_comparison_operator* parameter is used to allow a range of security service levels for a distribution. The combination of this parameter and the *security* parameter yields the permitted levels of security for the distribution.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *report_service_parms*, *routing_table_entry*, *service_parms*

Format: Enumeration

Possible Values
REQUIRE_LEVEL_GE

Selected_Row

Description: A *selected_row* is one of the rows that match the *row_selection_criteria.*

Verbs Supplied on: None

Verbs Returned on: List_DSU_Data

Sending_State

Description: The *sending_state* indicates the current state of a distribution being sent, from the DSU's perspective.

Verbs Supplied on: None

Verbs Returned on: Query_Distribution_Sending, Send_Distribution

Format: Enumeration

Value	Meaning
NOT_ASSIGNED	There has been no activity on the referenced distribution; the sequence number is a valid, unassigned value.
IN_TRANSIT	The agent has issued the sending verb for this distribution, and processing of the verb has not yet completed.
SPEC_SERVER_PENDING	The DSU has returned control to the agent after processing the sending verb, but specific server operations have not yet been completed.
COMMITTED	The DSU has accepted responsibility for the distribution after completing specific server operations (if any).
TERMINATED	The distribution could not be sent, and the agent has not yet issued Sending_Sequence_Completed on this distribution.
COMPLETED	The agent has issued Sending_Sequence_Completed. At this point, the DSU can stop tracking this distribution.

Seqno_To_Clean_Up

Description: The *sequence_number_to_clean_up* identifies a distribution which has been sent by the DSU, but for which the sending agent has not issued Sending_Sequence_Completed to acknowledge the transfer of responsibility.

Verbs Supplied on: None

Verbs Returned on: Send_Distribution

Format: Signed binary integer (1-origin)

Server

Description: The *server* is the name of the transaction program to be used to read the server object at the origin and to store the server object at the destination. Server name values are assigned according to the rules of SNA Transaction Program names.

Verbs Supplied on: Assign_Read_Access, Backout_Server_Object, Initiate_Read, Initiate_Write, Query_Last_Byte_Received, Read, Release_Read_Access, Send_Distribution, Terminate_Read, Terminate_Restartability, Terminate_Write, Write

Verbs Returned on: Get_Distribution_Info, Receive_Distribution

Subtables Found in: *server_list_entry*

Format: Character string, except for first byte

CGCSGID: 01134-00500 (Character Set AR)

String Conventions: Leading, imbedded, and trailing space (X′40′) characters are not allowed.

The first byte of an SNA-registered transaction program name ranges in value from X′00′ to X′3F′. When the first byte ranges in value from X′41′ to X′FF′, the transaction program is not SNA registered. X′40′ is not a valid first byte.

Server_Access

Description: The *server_access* parameter gives the server the information it needs to access the server object.

Verbs Supplied on: Assign_Read_Access, Backout_Server_Object, Initiate_Read, Release_Read_Access, Send_Distribution

Verbs Returned on: Get_Distribution_Info, Receive_Distribution, Terminate_Write

Presence Rule: May occur only if *specific_server_info* is absent.

Format: Undefined byte string

Server_Bytes_Received

Description: The *server_bytes_received* flag indicates whether an agent that issues List_Distributions_Being_Received is interested in how many bytes of the server object have been received for each distribution.

Verbs Supplied on: List_Distributions_Being_Received

Verbs Returned on: None

Format: Enumeration

Value	Meaning
YES	Indicate how many server object bytes have been received for each distribution.
NO	Do not indicate how many server object bytes have been received for each distribution.

Server_Bytes_Remaining

Description:	The *server_bytes_remaining* flag indicates whether an agent that issues List_Distributions_Being_Sent is interested in how many bytes of the server object remain to be sent for each distribution.
Verbs Supplied on:	List_Distributions_Being_Sent
Verbs Returned on:	None
Format:	Enumeration

Value	Meaning
YES	Indicate how many server object bytes remain to be sent for each distribution.
NO	Do not indicate how many server object bytes remain to be sent for each distribution.

Server_Instance_ID

Description:	The *server_instance_identifier* indicates which copy of the specified server is reading or writing a particular server object.
Verbs Supplied on:	Read, Terminate_Read, Terminate_Write, Write
Verbs Returned on:	Initiate_Read, Initiate_Write
Format:	Undefined byte string

Server_List_Entry

Description:	The *server_list_entry* describes one row in the Server List data structure.
Verbs Supplied on:	Add_DSU_Data, List_DSU_Data, Modify_DSU_Data, Remove_DSU_Data
Verbs Returned on:	List_DSU_Data

Server_Object_Byte_Count

Description: The *server_object_byte_count* is the number of bytes of all the segments of the server object. An FS2-capable DSU originating a distribution must either supply a correct byte count or omit the field completely; for FS1, the byte count need not be accurate.

Verbs Supplied on: Initiate_Write, Send_Distribution

Verbs Returned on: Get_Distribution_Info, Receive_Distribution

Format: Unsigned binary integer (1-origin); maximum: 2**64 - 2

Service_Parms

Description: The *service_parameters* describe the types and levels of service requested for the distribution. These parameters are provided by the sending agent.

Verbs Supplied on: List_Adjacent_DSUs, List_Connections, Reroute_Distribution_Copies, Send_Distribution

Verbs Returned on: Get_Distribution_Info, Receive_Distribution, Receive_Distribution_Report

Session_Reference

Description: The *session_reference* identifies the session for a log entry. It may be either a sending or receiving session for which the distribution or the exception is being logged.

Verbs Supplied on: None

Verbs Returned on: Get_Distribution_Log_Entry, Get_Exception_Log_Entry

Sibling

Description: The *sibling* is one of the ID (or T) values necessary to describe the detected condition. The structure identified by *sibling* is a child of the parent identified in *parent_spec* or a sibling of the structure identified in *structure_spec*. The class of the *sibling* structure is the same as *structure_class*. The expected position, when applicable, is given by *structure_position*.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *SNA_condition_report*

Presence Rule: Presence is governed by the *SNA_report_code*. The maximum number of occurrences is governed by *structure_class*. For LENGTH_BOUNDED_LT, the maximum number is 98; for LENGTH_BOUNDED_LLID, the maximum number is 49.

Format: Undefined byte string

SNA_Condition_Report

Description: The *SNA_condition_report* describes the condition being reported. The condition is always identified by an *SNA_report_code*.

Certain conditions can be more fully described by supplementary information. Conditions pertaining to one or more structures in a format can have the location and contents of each of those structures specified by a *structure_report*. Certain conditions arise from inconsistencies among multiple portions of the MU. Each portion is described by a separate *structure_report*. Other information related to the condition can be specified in a *supplemental_report*.

Verbs Supplied on: None

Verbs Returned on: Receive_Distribution_Report

SNA_Report_Code

Description: The *SNA_report_code* is an SNA-registered code identifying the condition that is being reported. Refer to Appendix E for allowable values and descriptions.

Verbs Supplied on: None

Verbs Returned on: Get_Exception_Log_Entry

Subtables Found in: *SNA_condition_report*

Format: Byte string, see *SNA Formats* for further description.

Specific_Server_Info

Description: The *specific_server_information* contains control information that is passed to a specific server to allow it to build a server object.

Verbs Supplied on: Backout_Server_Object, Initiate_Read, Send_Distribution

Verbs Returned on: Get_Distribution_Info, Receive_Distribution, Terminate_Write

Presence Rule: Never present when *server_access* is present.

Format: Undefined byte string

Specific_Server_Report

Description: The *specific_server_report* reports on the actions of a specific server. It is passed from the server to the agent by SNA/DS, either as a returned parameter on a distribution request verb, or on the stand-alone verb Obtain_Local_Server_Report. SNA/DS has no control over or knowledge of its contents.

Verbs Supplied on: Backout_Server_Object

Verbs Returned on: Backout_Server_Object, Get_Distribution_Info, Initiate_Read, Initiate_Write, Obtain_Local_Server_Report, Query_Distribution_Sending, Receive_Distribution, Send_Distribution, Terminate_Read, Terminate_Restartability, Terminate_Write

Format: Undefined byte string

Starting_Queue_Entry_ID

Description: The *starting_queue_entry_identifier* indicates the first queue entry identifier to be considered when processing a List_Queue_Entries verb.

Verbs Supplied on: List_Control_MU_Queue, List_Queue_Entries

Verbs Returned on: None

Format: Undefined byte string

Starting_Row_Number

Description: The *starting_row_number* indicates the first row of a DSU data structure that the DSU should consider in listing a data structure.

Verbs Supplied on: List_DSU_Data

Verbs Returned on: None

Format: Signed binary integer (1-origin)

Structure_Byte_Offset

Description: The *structure_byte_offset* marks the start of *structure_contents* within the reported-on structure. If *structure_segment_num* is present, this value is the offset from the start of the indicated segment; otherwise, it is the offset from the beginning of the structure.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *SNA_condition_report*

Format: Signed binary integer (1-origin)

Structure_Class

Description: The *structure_class* is the class of the reported-on structure and of any siblings identified in *sibling*. For more details on the classes of structures, see Appendix G.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *SNA_condition_report*

Format: Enumeration

Possible Values

LENGTH_BOUNDED_LLID
LENGTH_BOUNDED_LT
DELIMITED_LLID
DELIMITED_LT
IMPLIED_LLID
IMPLIED_LT

Structure_Contents

Description: The *structure_contents* is the portion of the MU that is relevant to the detected condition. Typically, the *structure_contents* will contain the header of the structure and at least the beginning of its contents. When the condition can be isolated to a portion of the structure, the *structure_contents* will contain only that portion of the structure relevant to the condition. In this case, the *structure_segment_num* and *structure_byte_offset* locate the portion of the structure relevant to the condition.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *SNA_condition_report*

Presence Rule: Allowed only when *structure_state* has the value PRESENT.

Format: Undefined byte string

Structure_ID_Or_T

Description: The *structure_ID_or_T* is the ID or T value of the structure. IDs are the registered GDS codepoints and are referred to in *SNA Formats*. T values are architecture-specific values relative to the encompassing ID.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *SNA_condition_report*

Presence Rule: Required except when the siblings contain all pertinent ID (or T) values. In this case, the structures specified by each *sibling* are the structures being reported on.

Format: Undefined byte string

Structure_Instance

Description: The *structure_instance* is used when the structure is one of multiple occurrences of a repeatable structure. The value of *structure_instance* identifies the particular instance within a position.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *SNA_condition_report*

Format: Signed binary integer (1-origin)

Structure_Position

Description: The *structure_position* is either the actual or expected position of this structure within its parent in this particular MU. Multiple consecutive instances of a repeatable structure share a single position; they can be distinguished by *structure_instance*.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *SNA_condition_report*

Format: Signed binary integer (1-origin)

Structure_Report

Description: The *structure_report* reports on a structure involved in a format-related condition. Depending on the condition, the *structure_report* may describe a structure that was present in or absent from the reported-on MU.

A format condition has its location in the MU pinpointed by a *structure_spec* and a list of *parent_spec* parameters that define a line-of-descent. The line-of-descent begins with the MU and continues down the parent-child hierarchy to a level as low as the particular condition warrants. A registered ID must appear in a *structure_report*; if the reported-on structure is not itself a registered ID, its line-of-descent must be traced up to include a registered ancestor.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *SNA_condition_report*

Presence Rule: Presence governed by the *SNA_report_code*.

Structure_Segment_Num

Description: The *structure_segment_number* is the segment of the structure in which the condition was detected.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *SNA_condition_report*

Presence Rule: Occurs when the beginning of *structure_contents* was not contained in the first segment of the reported-on structure.

Format: Signed binary integer (1-origin)

Structure_Spec

Description: The *structure_specification* contains the identifier (ID or T) and the class of a structure. For a structure that occurs multiple times, the instance may also be included. The value of the *structure_instance* identifies the particular instance. The position of this structure within its parent structure may also be included. This would typically be done when the parent structure contains unordered children.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *SNA_condition_report*

Presence Rule: Absent only when *structure_class* has the value LENGTH_BOUNDED_LT and the repeated values for *sibling* contain all the pertinent T values.

Structure_State

Description: The *structure_state* indicates whether the reported-on structure was present or absent.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *SNA_condition_report*

Format: Enumeration

Value	Meaning
ABSENT	The structure being reported on was absent.
PRESENT	The structure being reported on was present.

Supplemental_Report

Description: The *supplemental_report* contains other information pertaining to a condition. The contents of *supplemental_report* are governed by the *SNA_report_code*.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *SNA_condition_report*

Presence Rule: Presence governed by the *SNA_report_code*.

Format: Undefined byte string

Termination_Type

Description: The *termination_type* flag indicates the action the server is expected to take regarding the object it has just completed reading or writing.

Verbs Supplied on: Terminate_Read, Terminate_Write

Verbs Returned on: None

Format: Enumeration

Value	**Meaning**
NORMAL	The reading or writing of the server object completed normally; any saved checkpoints can be discarded.
SUSPEND	The reading or writing of the server object is being temporarily suspended. Any checkpoints that have been taken should be saved to allow Byte-Count Restart. This is the typical value for a Terminate_Read, since Byte-Count Restart may be required despite normal completion of send-side server operations.
ABORT	The reading or writing of the server object completed abnormally; any saved checkpoints can be discarded.

Unit_Of_Work_ID

Description: The *unit_of_work_identifier* identifies a unit of work for a server. It is assigned by the caller, either an agent or DS, on the Initiate_Write verb. The *unit_of_work_identifier* for DS is always the *distribution_identification*; for an agent, it can be any agent-specific byte string.

Verbs Supplied on: Initiate_Read, Initiate_Write, Release_Read_Access

Verbs Returned on: None

Year

Description: The *year* parameter gives the year portion of the date.

Verbs Supplied on: None

Verbs Returned on: None

Subtables Found in: *connection_definitions_entry*, *date*

Format: Signed binary integer (1-origin)

Appendix G. Encodings

Introduction

This appendix contains the format descriptions of the FS1 and FS2 message units. The format descriptions are comprised of two parts: *header description tables* and *structure descriptions*. A header description table contains the header information for each structure associated with a particular message unit. A structure description contains a prose description of the structure, bit-level representations, and any presence rules or length restrictions associated with a particular structure.

The definition of SNA/Distribution Services (DS) requires a byte-accurate description of the formats that must be understood by all DSUs. The DS formats are described in terms of encoded fields referred to as "structures" and the hierarchical relationship between these structures. In this appendix, the header description tables show each structure and its header. Elsewhere in this book, the header length is assumed not to be part of the overall structure length (e.g., *SNA_report_code*).

Structure Classifications

Fields and groupings of fields are known as structures. They are categorized in terms of their hierarchical position ("atomic," "child," or "parent"), the method by which their beginning and endings are determined, (length-bounded, delimited, or implied) and which kind of header is used to identify them (LT or LLID). Only certain combinations of characteristics are possible.

Length-bounded Structures

Length-bounded structures consist of a header and usually some following information. A header may be either two bytes in length, referred to as an "LT" (length and type), or four bytes in length, referred to as an "LLID" (length and GDS codepoint). In either case, the length bytes include the length of the header itself and the following information, if any. For FS1, a header may be either two bytes in length, referred to as an "LT," or five bytes in length, referred to as an "LLIDF" (length, GDS codepoint, and format byte).

Atomic Structures

In many cases, a structure consists only of its own header followed by data. These structures cannot be decomposed, and therefore they are called "atomic." Atomic structures are always length-bounded and may have either LT or LLID headers.

Parent and Child Structures

Structures can contain other structures within them. The containing structure is known as a *parent* structure and the contained structures are known as *children*. These terms are relative, since a nonatomic child structure itself contains other structures and is a parent to them. Children of the same parent are siblings of each other. Parent structures may be length-bounded, delimited, or implied; and may be identified by LTs or LLIDs.

Length-Bounded Parent Structures

In this case, the parent structure has its own header, either an LT or an LLID. Its length includes the lengths of all its children plus the length of its own header. A length-bounded parent exists both as a logical grouping of its children and as an explicit encoded structure at its own encoding level.

Delimited Parent Structures

Sometimes it is convenient to define a group of related structures as existing within a parent structure without having that parent structure appear as a length-bounded structure in the message. The beginning and end of the parent are defined by its first and last children. These children are known as delimiters, the first child is the prefix delimiter and the last is the suffix delimiter. Delimiter children are length-bounded and must be present. They may be null, that is, with an LT of length=2 or an LLID of length=4. When the children's headers are LTs, the parent is classified as a delimited LT structure. When they are LLIDs, the parent is a delimited LLID structure.

Implied Parent Structures

It is possible to define a set of related structures as children of a parent structure where the existence and boundaries of the parent are implied by the existence and order of certain child structures. This set of children may occur within the parent structure, either ordered or unordered, until a structure occurs that is not an element of this set. This break in sequence implies the boundary between parent structures. Depending on its children's headers, an implied parent is classified as either implied LT or implied LLID.

Segmented Structures

Length-bounded LLID Structures may be either segmentable or non-segmentable. For segmentable structures, the most significant bit of the LL bytes indicates whether any particular segment is the last (bit is equal to 0) or not last (bit is equal to 1) segment of the structure. The ID bytes of the segmentable structure are present on the first segment only.

For FS1, segmentation is indicated by the contents of the F byte (the fifth byte of the LLIDF header). Structures may be segmented when the most significant bit of the F byte is *on*. If the most significant bit is *on*, then three more bytes, the ISS bytes, follow the LLIDF header. The ISS bytes indicate whether a particular segment is the last segment of a structure. In each segment except the last segment of a structure, the I byte contains X'20'. In the last segment of a structure, the I byte contains X'00'. The SS bytes contain X'0000'.

Properties of Parent Structures

Order

A parent structure may have either ordered or unordered children. Ordered children occur in the parent structure in the same order as they are described in the format description table. Unordered children may occur in the parent structure in any order.

Unrecognized Children

Future enhancements to the formats might add structures that will not be recognized by implementations of the current format definitions. The current format must specify for each parent whether or not unrecognized child structures are allowed. If they are allowed, the definition must specify how long they might be. When unrecognized structures are found where they are allowed, they must be passed through without change at intermediate locations and gracefully ignored at final destinations. Unrecognized structures are identified by either LT or LLID headers, being of the same type as their siblings.

Number of Children

The number of children within a parent may range from a required minimum to an allowed maximum. For example, a parent might have several children, each defined with an occurrence of 0-1, and a number of children defined as 1. This means that any one, but only one, child is allowed.

Header Description Table

The header information and primary syntax associated with each structure are formally described in tabular form. These header description tables represent the formatting information required to either parse or build DS structures.

Structure Name

The first column of the header description table identifies DS structures, by name, and illustrates their hierarchical relationship by indentation of the column entries. The order of the structure entries in the table represents, unless specified otherwise, the order in which the structures appear in a DS message unit.

Structure Reference (Struct Ref)

As header information and primary syntax are described in the header description of a particular table, the semantics, bit representations, presence rules, and other characteristics are described formally in the structure description. This column contains a reference page number to where this structure information is found.

Structure Class (Struct Class)

Structures are classified as either length-bounded LLIDs (ID), length-bounded LTs (T), delimited LLIDs (Del-ID), delimited LTs (Del-T), implied LLIDs (Imp-ID), or implied LTs (Imp-T).

A structure classified as delimited must contain at least two required, length-bounded children that act as the prefix (pfx) and suffix (sfx) of the delimited structure. The "/pfx" notation indicates the length-bounded child structure that serves as the prefix for its parent delimited structure. The "/sfx" notation indicates the length-bounded structure that serves as the suffix for its parent delimited structure.

A structure classified as implied uses an identified child to identify the beginning of a sequence of children. The "/idc" notation indicates the length-bounded structure that serves as an identified child of its parent implied structure.

The same notation is applied to the Format Set 1 encodings. Structures in FS1 are classified as either length-bounded LLIDFs (IDF), length-bounded LTs (T), delimited LLIDFs (Del-IDF), delimited LTs (Del-T), implied LLIDFs (Imp-IDF), or implied LTs (Imp-T).

The "/seg" notation indicates that segmentation is allowed.

ID/T

This column contains the ID or T value within the header, in hexadecimal. To indicate that a delimited structure is identified by its prefix, the notation "pfx" is used. To indicate that an implied structure is identified by one of its children, the notation "idc," for identified child, is used.

Length

This column describes the length verification that would be appropriate at presentation services time. The range of length values specifies the minimum and maximum lengths of structures that an implementation is required to receive. For structures that allow unrecognized children, the maximum length value accommodates the possibility of these yet-to-be-defined structures. On the sending side, the maximum length value for a particular structure may be determined by subtracting the unrecognized reserve, if unrecognized children are allowed, from the maximum length.

Note: An asterisk denotes length restrictions for a particular structure. Length restrictions are detailed in the corresponding structure description.

Occurrences

Multiple occurrences of DS structures may or may not be permitted. A value of "1 - <some number>" in this column indicates the allowed range of occurrences of the corresponding structure. A value of "≥1" indicates that there is no architecturally defined maximum. A value of "1" in this column indicates that only a single instance of the corresponding structure is appropriate. A value of "0 - 1" indicates that an instance of the corresponding structure is optional.

Note: An asterisk denotes presence rules for a particular structure. Presence rules are detailed in the corresponding structure description.

Children

Unrecognized Children Allowed (Unrec): An entry of "Y" in the "Unrec" column indicates that the corresponding structure tolerates unrecognized child structures. An entry of "N" indicates that the particular structure tolerates only the architecturally-defined child structures. An entry of "—" indicates that unrecognized children are not applicable to the particular structure. By definition, atomic structures do not contain children, recognized or not.

Order: A value of "Y" in this column indicates that children are ordered, a value of "N" indicates that children are unordered, and a value of "—" indicates that no children are present.

Note: If a structure is atomic, this column is not applicable.

Number (Num): Each parent structure contains a certain number of different children. This column specifies the minimum and maximum number of different children for a particular parent structure. The maximum number also accounts for unrecognized children, if they are allowed within the parent structure. This column does not account for multiple occurrences of a particular child structure within the parent structure. The number of occurrences of each child is indicated in the ″Occurrences″ column.

Subtable: Sometimes the need to divide large tables into subtables becomes apparent, particularly when common children appear frequently within different header description tables. This column contains a reference page number to where these common children are described.

Structure Description

The structure description is referenced by a page number appearing in the ″Structure Reference″ column corresponding to each structure in the header description table. This description contains information pertaining to the data portion of a particular structure. Prose descriptions, presence rules, and semantics associated with the corresponding entry in the header description table may appear in the structure description.

Header Description Tables for FS2 Message Units

DISTRIBUTION TRANSPORT MESSAGE UNIT (DTMU)

Table 69 (Page 1 of 2). Distribution Transport Message Unit

Structure Name	Struct Ref Pg	Struct Class	ID/T	Length	Occur-rences	Children			
						Unrec	Order	Num	Sub Table
Dist_Transport_MU	568	Del-ID	pfx	≥53*	1	Y	Y	4-12	—
Transport_Prefix	568	ID/pfx	1570	8-18	1	N	Y	1-3	—
Hop_Count	568	T	01	4	1	—	—	—	—
MU_ID	568	T	03	6	0-1*	—	—	—	—
MU_Instance_Number	568	T	06	4	0-1*	—	—	—	—
Transport_Command	569	ID/seg	1571	29-4096*	1	Y	Y	3-30	—
Dist_Flags	569	T	01	5	0-1	—	—	—	—
Service_Parms	570	T	02	5-32	0-1	—	—	—	—
Server_Obj_Byte_Count	573	T	03	10	0-1*	—	—	—	—
Origin_Agent	573	T	04	3-10	1	—	—	—	—
Server	573	T	05	3-10	0-1*	—	—	—	—
Origin_DSU	573	T	06	8-22	1	N	Y	2	—
Origin_RGN	574	T	01	3-10	1	—	—	—	—
Origin_REN	574	T	02	3-10	1	—	—	—	—
Origin_User	574	T	07	8-22	0-1	N	Y	2	—
Origin_DGN	574	T	01	3-10	1	—	—	—	—
Origin_DEN	575	T	02	3-10	1	—	—	—	—
Seqno_DTM	575	T	08	14-17*	1	—	—	—	—
Supplemental_Dist_Info1	576	T	09	3-10	0-1	—	—	—	—
Agent_Correl	577	T	0A	3-130	0-1	—	—	—	—
Report-To_DSU	577	T	0B	8-22	0-1	N	Y	2	—
Report-To_RGN	577	T	01	3-10	1	—	—	—	—
Report-To_REN	577	T	02	3-10	1	—	—	—	—
Report-To_User	578	T	0C	8-22	0-1	N	Y	2	—
Report-To_DGN	578	T	01	3-10	1	—	—	—	—
Report-To_DEN	578	T	02	3-10	1	—	—	—	—
Report_Service_Parms	579	T	0D	5-32	0-1	—	—	—	—
Report-To_Agent	581	T	0E	3-10	0-1	—	—	—	—
Dest_Agent	582	T	0F	3-10	0-1	—	—	—	—
Unrecognized_Reserve	605	T	—	2-3728	—	—	—	—	—
Dest_List	582	ID/seg	1572	12-11268	1	N	Y	1	—
Dest	582	Imp-T	idc	8-5654	≥1	N	Y	1-2	—
Dest_DSU	582	T/idc	01	8-22	1	N	Y	2	—
Dest_RGN	583	T	01	3-10	1	—	—	—	—

Table 69 (Page 2 of 2). Distribution Transport Message Unit

Structure Name	Struct Ref Pg	Struct Class	ID/T	Length	Occur-rences	Children			
						Unrec	Order	Num	Sub Table
Dest_REN	583	T	02	3-10	1	—	—	—	—
Dest_User	583	T	02	8-22	≥0	N	Y	2	—
Dest_DGN	583	T	01	3-10	1	—	—	—	—
Dest_DEN	584	T	02	3-10	1	—	—	—	—
Agent_Object	584	ID/seg	1573	5-32767	0-1	—	—	—	—
Server_Object	584	ID/seg	1574	≥5	0-1	—	—	—	—
Supplemental_Dist_Info2	584	ID/seg	1580	5-32767	0-1	—	—	—	—
Unrecognized_Reserve	605	ID/seg	—	4-32767	—	—	—	—	—
DS_Suffix	584	ID/sfx	157F	4	1	—	—	—	—

Note: * Refer to FS2 Structure Descriptions starting on page 568 for presence rules and length restrictions.

DISTRIBUTION REPORT MESSAGE UNIT (DRMU)

Table 70 (Page 1 of 2). Distribution Report Message Unit

Structure Name	Struct Ref Pg	Struct Class	ID/T	Length	Occur-rences	Children			
						Unrec	Order	Num	Sub Table
Dist_Report_MU	585	Del-ID	pfx	≥77*	1	Y	Y	6-12	—
Report_Prefix	585	ID/pfx	157C	8-18	1	N	Y	1-3	—
Hop_Count	568	T	01	4	1	—	—	—	—
MU_ID	568	T	03	6	0-1	—	—	—	—
MU_Instance_Number	568	T	06	4	0-1*	—	—	—	—
Report_Command	585	ID/seg	1575	25-4096*	1	Y	Y	3-20	—
Service_Parms	570	T	02	5-32	0-1	—	—	—	—
Report-To_Agent	581	T	04	3-10	1	—	—	—	—
Reporting_DSU	585	T	06	8-22	1	N	Y	2	—
Reporting_RGN	585	T	01	3-10	1	—	—	—	—
Reporting_REN	585	T	02	3-10	1	—	—	—	—
Report_DTM	586	T	09	10-13*	1	—	—	—	—
Unrecognized_Reserve	605	T	—	2-4015	—	—	—	—	—
Report-To_DSU_User	587	ID	1583	12-48	1	N	Y	1-2	—
Report-To_DSU	577	T	01	8-22	1	N	Y	2	—
Report-To_RGN	577	T	01	3-10	1	—	—	—	—
Report-To_REN	577	T	02	3-10	1	—	—	—	—
Report-To_User	578	T	02	8-22	0-1	N	Y	2	—
Report-To_DGN	578	T	01	3-10	1	—	—	—	—
Report-To_DEN	578	T	02	3-10	1	—	—	—	—
Report_Information	588	ID/seg	1576	18-4096	1	Y	Y	1-24	—
Reported-On_Origin_DSU	588	T	06	8-22	0-1*	N	Y	2	—
Reported-On_Origin_RGN	588	T	01	3-10	1	—	—	—	—
Reported-On_Origin_REN	588	T	02	3-10	1	—	—	—	—
Reported-On_Origin_User	589	T	07	8-22	0-1*	N	Y	2	—
Reported-On_Origin_DGN	589	T	01	3-10	1	—	—	—	—
Reported-On_Origin_DEN	589	T	02	3-10	1	—	—	—	—
Reported-On_Seqno_DTM	590	T	08	14-17	1	—	—	—	—
Reported-On_Supp_Dist_Info1	592	T	09	3-10	0-1*	—	—	—	—
Reported-On_Agent_Correl	592	T	0A	3-130	0-1	—	—	—	—
Reported-On_Origin_Agent	592	T	0B	3-10	0-1*	—	—	—	—
Reported-On_Dest_Agent	592	T	0C	3-10	0-1*	—	—	—	—
Receiving_DSU	601	T	10	8-22	0-1	N	Y	2	—
Receiving_RGN	601	T	01	3-10	1	—	—	—	—
Receiving_REN	602	T	02	3-10	1	—	—	—	—
Unrecognized_Reserve	605	T	—	2-3849	—	—	—	—	—
SNA_Condition_Report	593	ID/seg	1532	10-32767	1	Y	Y	1-10	564
Reported-On_Supp_Dist_Info2	593	ID/seg	1582	5-32767	0-1*	—	—	—	—
Unrecognized_Reserve	605	ID/seg	—	4-32767	—	—	—	—	—

Table 70 (Page 2 of 2). Distribution Report Message Unit

Structure Name	Struct Ref Pg	Struct Class	ID/T	Length	Occurrences	Children			
						Unrec	Order	Num	Sub Table
DS_Suffix	584	ID/sfx	157F	4	1	—	—	—	—

Note: * Refer to FS2 Structure Descriptions starting on page 568 for presence rules and length restrictions.

DISTRIBUTION CONTINUATION MESSAGE UNIT (DCMU)

Table 71. Distribution Continuation Message Unit

Structure Name	Struct Ref Pg	Struct Class	ID/T	Length	Occurrences	Children			
						Unrec	Order	Num	Sub Table
Dist_Continuation_MU	593	Del-ID	pfx	≥18	1	Y	Y	2-10	—
Continuation_Prefix	593	ID/pfx	157B	14-24	1	N	Y	2-3	—
MU_ID	568	T	03	6	1	—	—	—	—
MU_Instance_Number	568	T	06	4	1	—	—	—	—
Restarting_Byte_Position	593	T	02	10	0-1	—	—	—	—
Agent_Object	584	ID/seg	1573	5-32767	0-1	—	—	—	—
Server_Object	584	ID/seg	1574	≥5	0-1	—	—	—	—
Supplemental_Dist_Info2	576	ID/seg	1580	5-32767	0-1*	—	—	—	—
Unrecognized_Reserve	605	ID/seg	—	4-32767	—	—	—	—	—
DS_Suffix	584	ID/sfx	157F	4	1	—	—	—	—

Note: * Refer to FS2 Structure Descriptions starting on page 568 for presence rules.

SNA CONDITION REPORT

Table 72. SNA Condition Report

Structure Name	Struct Ref Pg	Struct Class	ID/T	Length	Occur-rences	Children			
						Unrec	Order	Num	Sub Table
SNA_Condition_Report	593	ID	1532	10-32767	1	Y	Y	1-10	—
SNA_Report_Code	594	T	7D	6	1	—	—	—	—
Structure_Report	594	T	01	14-255	0-10*	Y	Y	2-10	—
Structure_State	594	T	01	3	1	—	—	—	—
Structure_Contents	595	T	02	3-100	0-1*	—	—	—	—
Parent_Spec	595	T	03	5-17	0-7	N	Y	1-4	—
Parent_ID_Or_T	595	T	01	3-4	1	—	—	—	—
Parent_Class	595	T	02	3	0-1*	—	—	—	—
Parent_Position	596	T	03	4	0-1	—	—	—	—
Parent_Instance	596	T	04	4	0-1	—	—	—	—
Structure_Spec	596	T	04	5-17	0-1*	N	Y	1-4	—
Structure_ID_Or_T	596	T	01	3-4	0-1*	—	—	—	—
Structure_Class	597	T	02	3	0-1*	—	—	—	—
Structure_Position	597	T	03	4	0-1	—	—	—	—
Structure_Instance	597	T	04	4	0-1	—	—	—	—
Structure_Segment_Number	597	T	05	4	0-1*	—	—	—	—
Structure_Byte_Offset	598	T	06	4	0-1	—	—	—	—
Sibling_List	598	T	07	3-100	0-1*	—	—	—	—
Unrecognized_Reserve	605	T	—	2-241	—	—	—	—	—
Reported-On_Dest_List	598	Del-T	pfx	12-11268	0-1*	N	Y	3	—
Reported-On_Dest_Prefix	598	T/pfx	08	2	1	—	—	—	—
Reported-On_Dest	598	Imp/T	idc	8-5654	≥1	N	Y	1-2	—
Reported-On_Dest_DSU	598	T/idc	09	2-22	1	N	Y	0-2	—
Reported-On_Dest_RGN	599	T	01	3-10	0-1*	—	—	—	—
Reported-On_Dest_REN	599	T	02	3-10	0-1*	—	—	—	—
Reported-On_Dest_User	599	T	0A	8-22	≥0	N	Y	2	—
Reported-On_Dest_DGN	600	T	01	3-10	1	—	—	—	—
Reported-On_Dest_DEN	600	T	02	3-10	1	—	—	—	—
Reported-On_Dest_Suffix	600	T	0B	2	1	—	—	—	—
Supplemental_Report	600	T	03	3-255	0-5*	—	—	—	—
Unrecognized_Reserve	605	T	—	2-17664	—	—	—	—	—

Note: * Refer to FS2 Structure Descriptions starting on page 568 for presence rules and length restrictions.

SENDER EXCEPTION MESSAGE UNIT (SEMU)

Table 73. Sender Exception Message Unit

Structure Name	Struct Ref Pg	Struct Class	ID/T	Length	Occur-rences	Children			
						Unrec	Order	Num	Sub Table
Sender_Exception_MU	601	ID	1578	10-256	1	Y	Y	1-10	—
SNA_Report_Code	594	T	7D	6	1	—	—	—	—
MU_ID	568	T	03	6	0-1	—	—	—	—
MU_Instance_Number	568	T	06	4	0-1*	—	—	—	—
Unrecognized_Reserve	605	T	—	2-236	—	—	—	—	—

Note: * Refer to FS2 Structure Descriptions starting on page 568 for presence rules.

RECEIVER EXCEPTION MESSAGE UNIT (REMU)

Table 74. Receiver Exception Message Unit

Structure Name	Struct Ref Pg	Struct Class	ID/T	Length	Occur-rences	Children			
						Unrec	Order	Num	Sub Table
Receiver_Exception_MU	601	Del-ID	pfx	≥25	1	Y	Y	2-10	—
Receiver_Exception_Command	601	ID/pfx	1577	15-512	1	Y	Y	2-8	—
Sender_Retry_Action	601	T	01	3	1	—	—	—	—
MU_ID	568	T	03	6	0-1	—	—	—	—
MU_Instance_Number	568	T	06	4	0-1*	—	—	—	—
Receiving_DSU	601	T	16	8-22	1	N	Y	2	—
Receiving_RGN	601	T	01	3-10	1	—	—	—	—
Receiving_REN	602	T	02	3-10	1	—	—	—	—
Unrecognized_Reserve	605	T	—	2-473	—	—	—	—	—
Unrecognized_Reserve	605	ID	—	≥4	—	—	—	—	—
SNA_Condition_Report	593	ID/sfx	1532	10-1024	1	Y	Y	1-10	564

Note: * Refer to FS2 Structure Descriptions starting on page 568 for presence rules.

COMPLETION QUERY MESSAGE UNIT (CQMU)

Table 75. Completion Query Message Unit

Structure Name	Struct Ref Pg	Struct Class	ID/T	Length	Occur-rences	Children			
						Unrec	Order	Num	Sub Table
Completion_Query_MU	602	ID	1579	14-256	1	Y	Y	2-10	—
MU_ID	568	T	03	6	1	—	—	—	—
MU_Instance_Number	568	T	06	4	1	—	—	—	—
Unrecognized_Reserve	605	T	—	2-242	—	—	—	—	—

COMPLETION REPORT MESSAGE UNIT (CRMU)

Table 76. Completion Report Message Unit

Structure Name	Struct Ref Pg	Struct Class	ID/T	Length	Occur-rences	Children			
						Unrec	Order	Num	Sub Table
Completion_Report_MU	602	ID	157A	7-256	1	Y	Y	1-10	—
Indicator_Flags	602	T	01	3	1	—	—	—	—
MU_ID	568	T	03	6	0-1	—	—	—	—
MU_Instance_Number	568	T	06	4	0-1*	—	—	—	—
Last_Structure_Received	603	T	04	4	0-1*	—	—	—	—
Last_Byte_Received	603	T	05	10	0-1*	—	—	—	—
Unrecognized_Reserve	605	T	—	2-225	—	—	—	—	—

Note: * Refer to FS2 Structure Descriptions starting on page 568 for presence rules.

PURGE REPORT MESSAGE UNIT (PRMU)

Table 77. Purge Report Message Unit

Structure Name	Struct Ref Pg	Struct Class	ID/T	Length	Occur-rences	Children			
						Unrec	Order	Num	Sub Table
Purge_Report_MU	603	ID	157E	10-256	1	Y	Y	1-10	—
MU_ID	568	T	03	6	1	—	—	—	—
Unrecognized_Reserve	605	T	—	2-246	—	—	—	—	—

RESET REQUEST MESSAGE UNIT (RRMU)

Table 78. Reset Request Message Unit

Structure Name	Struct Ref Pg	Struct Class	ID/T	Length	Occurrences	Children			
						Unrec	Order	Num	Sub Table
Reset_Request_MU	603	ID	1585	21-23	1	N	Y	2	—
MU_ID	568	T	03	6	1	—	—	—	—
Reset_DTM	603	T	09	11-13	1	—	—	—	—

RESET ACCEPTED MESSAGE UNIT (RAMU)

Table 79. Reset Accepted Message Unit

Structure Name	Struct Ref Pg	Struct Class	ID/T	Length	Occurrences	Children			
						Unrec	Order	Num	Sub Table
Reset_Accepted_MU	604	ID	1586	21-23	1	N	Y	2	—
MU_ID	568	T	03	6	1	—	—	—	—
Reset_DTM	603	T	09	11-13	1	—	—	—	—

FS2 Structure Descriptions

Dist_Transport_MU

Description: The *distribution_transport_message_unit* transports agent and/or server objects for distribution to one or more users or application programs.

Length Restriction: The minimum length of a *dist_transport_MU* originated by an FS2 DSU is 54 bytes. This is due to the length restriction on the *Seqno_DTM*.

Transport_Prefix

Description: The *transport_prefix* identifies the beginning of the *dist_transport_MU*. This structure carries information that changes from DSU to DSU.

Hop_Count

Description: The *hop_count* is the remaining number of hops that may be traversed by a DS distribution on its way toward its destination DSUs. The *hop_count* is set by the origin DSU in the DTMUs and by the reporting DSUs for the DRMUs. The *hop_count* is decremented by 1 in every DSU through which the distribution passes. If the *hop_count* reaches 0 at an intermediate DSU, exception processing is invoked.

Format: Signed binary integer (1-origin)

MU_ID

Description: The *message_unit_identifier* is a number that uniquely identifies a distribution MU throughout its existence. An MU exists for only one hop, from one DSU to the adjacent DSU. In REMUs and SEMUs, the *MU_ID* refers to a distribution MU. An *MU_ID* is unique only for a particular *LU name*, *mode name* combination.

Presence Rule: If the *MU_ID* is absent, exception reporting may not be requested.

Format: Signed binary integer (1-origin)

MU_Instance_Number

Description: The *message_unit_instance_number* identifies the instance of a particular distribution message unit and its corresponding *MU_ID*.

Presence Rule: Precluded if an *MU_ID* is not present; otherwise, required.

Format: Signed binary integer (1-origin)

Transport_Command

Description: The *transport_command* contains the control information used by the distribution service to transport the distribution.

Length Restriction: The minimum length of a *transport_command* originated by an FS2 DSU is 30 bytes. This is due to the length restriction on the *Seqno_DTM*.

Dist_Flags

Description: The *distribution_flags* indicate services requested by the origin agent.

Note: If exception reporting is requested, the *MU_ID* is always present.

Format: Bit string

Byte	Bit	Content
0-1		LT header
2		Flags (bits 0-7) that must be understood and honored by all DSUs
	0	Exception report flag indicating whether an exception report is to be sent if the distribution is aborted: 0 no exception report to be sent (default) 1 exception report to be sent
	1-7	Reserved
3		Flags (bits 0-7) that must be understood and honored by destination DSUs, but that can be ignored by intermediate DSUs
	0-7	Reserved
4		Flags (bits 0-7) that are ignored by DSUs if not understood
	0-7	Reserved

Service_Parms

Description: The *service_parameters* structure describes the types and levels of service requested for the distribution. The parameters in this structure are provided by the origin agent. The *service_parameters* used in the DTMU and the DRMU are similar; the differences in such usage and the default values used for absent *service_parameter* (SP) triplets are discussed under the individual triplets below. Refer to Appendix C for details on the base and option subsets for *service_parameters*. The default values specified below are assumed for absent *service_parameter* (SP) triplets.

In FS1, the *service_parameters* are specified by the origin agent in Dist_MU *type* TRANSPORT. The specification for deriving the *service_parameters* for Dist_MU *type* REPORT is found in the description of *report_service_parameters* on page 579.

Format: Special format consisting of ordered, optional, SP triplets of the following general structure:

Byte	**Bit**	**Content**
0		Parameter type: All parameter type byte values are defined by or reserved for SNA/DS.
1		Comparison operator:
	0-3	1100 REQUIRE_LEVEL_GE 1110 REQUIRE_SUPPORT_FOR Note: All other values for bits 0-3 are reserved.
	4-7	Reserved
2		Value: The meaning of this byte depends on the parameter type.

Byte	**Content**
0-1	LT header
2-31	Up to 10 different *service_parameter* (SP) triplets may be carried in one distribution. Each triplet, when present, appears in ascending sequence of parameter type. For FS2, the capacity triplet is not used in the DRMU. For FS1, the capacity triplet is used. For FS2, all service parameters are optional in both the DTMU and the DRMU. For FS1, the first three parameters are present in both Dist_MU *types* TRANSPORT and REPORT. The architecturally defined service parameters are given below:

Priority SP Triplet

Byte	Content		
0	X'01'		
1	X'C0'	REQUIRE_LEVEL_GE	
2	X'F0'	FAST	(default)
	X'D0'	CONTROL	
	X'80'	DATA_16	(can be treated as DATAHI)
	X'78'	DATA_15	(can be treated as DATAHI)
	X'70'	DATA_14	(can be treated as DATAHI)
	X'68'	DATA_13	(can be treated as DATAHI)
	X'60'	DATA_12	(DATAHI)
	X'58'	DATA_11	(can be treated as DATAHI)
	X'50'	DATA_10	(can be treated as DATAHI)
	X'48'	DATA_9	(can be treated as DATAHI)
	X'40'	DATA_8	(can be treated as DATALO)
	X'38'	DATA_7	(can be treated as DATALO)
	X'30'	DATA_6	(can be treated as DATALO)
	X'28'	DATA_5	(can be treated as DATALO)
	X'20'	DATA_4	(DATALO)
	X'18'	DATA_3	(can be treated as DATALO)
	X'10'	DATA_2	(can be treated as DATALO)
	X'08'	DATA_1	(can be treated as DATALO)
	Note: All other values are reserved.		

Protection SP Triplet

Byte	Content	
0	X'02'	
1	X'C0'	REQUIRE_LEVEL_GE
2	X'10'	LEVEL1 (default when Priority SP is GE X'E0'): safe store may be performed.
	X'30'	LEVEL2 (default when Priority SP is LT X'E0'): safe store must be performed.
	Note: All other values are reserved.	

Capacity SP Triplet

Byte	Content
0	X'03'
1	X'C0' REQUIRE_LEVEL_GE
2	Capacity value is the exponent of the power of 2 that represents the value of the required capacity for the *server_object* in the DTMU:

X'00'	ZERO	(default when Priority SP is GE X'E0') used if there is no *server_object* in *dist_transport_MU*.
X'14'	1MB	one megabyte
X'16'	4MB	
X'18'	16MB	(default when Priority SP is LT X'E0')

Note: All other values are reserved.

1. In FS2, the Capacity SP triplet occurs only in a DTMU.

2. Receiving FS2 DSUs are always able to receive a capacity level of INDEFINITE (designated by X'E0FF' in bytes 1-2).
   Originating FS2 DSUs never generate the capacity level of INDEFINITE. The level replacing INDEFINITE is 16MB (X'C018').

3. The capacity requirement is for the *server_object*, and does not include the capacity needed to store and handle the other structures of the DTMU.

4. Implementations may accept other capacity levels as long as they can route the distribution responsibly.

Security SP Triplet

Byte	Content
0	X'04'
1	X'C0' REQUIRE_LEVEL_GE
2	X'01' LEVEL1 (default): security is not required. X'20' LEVEL2: security is required. Note: All other values are reserved.

Server_Obj_Byte_Count

Description: The *server_object_byte_count* is the number of bytes of all the segments of the *server_object*. An FS2-capable DSU originating a distribution either supplies a correct byte count, or omits the field completely; for FS1, the byte count need not be accurate.

Presence Rule: Optional when the *server_object* is present; otherwise, precluded.

Format: Unsigned binary integer (1-origin)

Origin_Agent

Description: The *origin_agent* is the transaction program at the DSU at which the distribution originated.

Format: Character string, except for first byte

CGCSGID: 01134-00500 (character set AR)

String Conventions: Leading, imbedded, and trailing space (X′40′) characters are not allowed.

The first byte of an SNA-registered transaction program name ranges in value from X′00′ to X′3F′. When the first byte ranges in value from X′41′ to X′FF′, the transaction program is not SNA registered. X′40′ is not a valid first byte.

Server

Description: The *server* is the name to be used to store the *server_object* at the destination.

Presence Rule: In FS2, optional when the *server_object* is present; otherwise, precluded. If optional and absent, the general server TP name is the default. In FS1, required when the *server_object* is present.

Format: Character string, except for first byte

CGCSGID: 01134-00500 (character set AR)

String Conventions: Leading, imbedded, and trailing space (X′40′) characters are not allowed.

The first byte of an SNA-registered transaction program name ranges in value from X′00′ to X′3F′. When the first byte ranges in value from X′41′ to X′FF′, the transaction program is not SNA registered. X′40′ is not a valid first byte.

Origin_DSU

Description: The *origin_DSU* is the name of the DSU at which the distribution originated.

Origin_RGN

Description: The *origin_RGN* is the first part of the name of the DSU at which the distribution originated. This is typically, but not necessarily, the network ID.

Format: Character string

CGCSGID: 01134-00500 (character set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

Origin_REN

Description: The *origin_REN* is the second part of the name of the DSU at which the distribution originated. This is typically, but not necessarily, the LU name.

Format: Character string

CGCSGID: 01134-00500 (character set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

Origin_User

Description: The *origin_user* is the user name of the originator of the distribution.

Origin_DGN

Description: The *origin_DGN* is the first part of the user name of the distribution originator.

Note: For FS1, when the Dist_MU is of type REPORT and the distribution report was generated by DS, null user names will occur.

Format: Character string

CGCSGIDs: 01134-00500 (base), 00930-00500 (enhanced character set)

String Conventions:

Base Leading, imbedded, and trailing space (X'40') characters are not allowed.

ECS Leading space (X'40') characters are not allowed, trailing space characters are not significant, and imbedded space characters are significant.

Origin_DEN

Description: The *origin_DEN* is the second part of the user name of the distribution originator.

Note: For FS1, when the Dist_MU is of type REPORT and the distribution report was generated by DS, null user names will occur.

Format: Character string

CGCSGIDs: 01134-00500 (base), 00930-00500 (enhanced character set)

String Conventions:

Base Leading, imbedded, and trailing space (X'40') characters are not allowed.

ECS Leading space (X'40') characters are not allowed, trailing space characters are not significant, and imbedded space characters are significant.

Seqno_DTM

Description: The *sequence_number/date-time*, in combination with the *origin_agent*, *origin_user*, and *origin_DSU*, uniquely identifies the distribution. The sequence number is the number assigned to the distribution by the origin agent. For FS2, the number ranges from 1 to (2**31)-1. For FS1, the number ranges from 0 (for report MUs) to 9999. Refer to Appendix D for migration details. The date of the distribution is assigned by the origin agent; the time of the distribution is assigned by the origin DSU. The offset from GMT for local time is included.

Note: FS2 tolerates sequence numbers with value 0 in message units that had, at some point, come from an FS1 network and had already specified a sequence number of 0 (i.e., DIA application status reports). However, sequence numbers with value 0 are never originated from within an FS2 network.

Length Restriction: Originating FS2 DSUs generate a GMT-based time. The minimum length for *seqno_DTM* is therefore 15 (1-origin).

Format: Byte string

Byte	Content
0-1	LT header
	SEQNO
2-5	Signed binary integer limited to (2**31)-1
	DATE
6-7	Year, in binary (e.g., 1989 is encoded as X′07C5′)
8	Month of the year, in binary (values from 1 to 12 are valid)
9	Day of the month, in binary (values from 1 to 31 are valid)
	TIME
10	Hour of the day, in binary (values from 0 to 23 are valid)
11	Minute of the hour, in binary (values from 0 to 59 are valid)
12	Second of the minute, in binary (values from 0 to 59 are valid)
13	Hundredth of the second, in binary (values from 0 to 99 are valid)
	GMT FLAGS
14	Indicates that specified TIME is GMT and identifies whether offsets from GMT are required to calculate local time. (Equivalent EBCDIC characters are shown in parentheses.) X′E9′ (z) no offset required X′4E′ (+) add required offset to GMT to get local time X′60′ (-) subtract required offset from GMT to get local time Note: All other values are reserved.
15	Hour offset from GMT, in binary, occurs when GMT flag ≠ X′E9′ (values from 0 to 23 are valid)
16	Minute offset from GMT, in binary, occurs when GMT flag ≠ X′E9′ (values from 0 to 59 are valid)

Examples:

A 9-byte date-time encoding is a date-time followed immediately by an EBCDIC ″Z″ and is considered to be GMT. Thus, 12:00GMT on 2 January 1988 would be

```
X'07C401020C000000E9'
  yyyyMMddHHmmsshhZ
```

An 11-byte date-time encoding is a date-time followed immediately by an EBCDIC ″+″ or ″-″ and two 1-byte binary numbers, and is considered to be GMT and the offset from GMT to local time. Thus, 7:00am on 2 January 1988 in New York would be 12:00GMT - 5 hours, or

```
X'07C401020C000000600500'
  yyyyMMddHHmmsshh- HHmm
```

Supplemental_Dist_Info1

Description: The *supplemental_dist_info1* structure is reserved for future use.

Format: Character string

Agent_Correl

Description: The *agent_correlation* is a string supplied by the origin agent. DS is not aware of its contents.

Format: Undefined byte string

Report-To_DSU

Description: The *report-to_DSU* is the name of the DSU to which distribution reports are to be sent. If both *report-to_DSU* and *report-to_user* are absent in the DTMU, the values generated in the DRMU for these structures default to the origin. If only *report-to_DSU* is present in the DTMU, then any report is sent to that DSU. If only *report-to_user* is present in the DTMU, then the reporting DSU will refer to its directory to determine *report-to_DSU*. For FS1, this information is valid only if Dist_MU is of type TRANSPORT.

Report-To_RGN

Description: The *report-to_RGN* is the first part of the DSU name to which distribution reports are to be sent. For FS1, this information is valid only if Dist_MU is of type TRANSPORT. This is typically, but not necessarily, the network ID.

Format: Character string

CGCSGID: 01134-00500 (character set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

Report-To_REN

Description: The *report-to_REN* is the second part of the DSU name to which distribution reports are to be sent. For FS1, this information is valid only if Dist_MU is of type TRANSPORT. This is typically, but not necessarily, the LU name.

Format: Character string

CGCSGID: 01134-00500 (character set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

Note: If a product chooses to implement DGN=REN, the enhanced character set (ECS) subset is implemented in a particular network, and any DGN contains an ECS character that is not an element of character set AR, then ECS characters may occur in this structure.

Report-To_User

Description: The *report-to_user* is the name of the user to which distribution reports are to be sent. If both *report-to_user* and *report-to_DSU* are absent in the DTMU, the values generated in the DRMU for these structures default to the origin. If only *report-to_user* is present in the DTMU the reporting DSU refers to its directory to determine *report-to_DSU*. For FS1, this information is valid only if Dist_MU is of type TRANSPORT.

Report-To_DGN

Description: The *report-to_DGN* is the first part of the user name to which distribution reports are to be sent. For FS1, this information is valid only if Dist_MU is of type TRANSPORT.

Format: Character string

CGCSGIDs: 01134-00500 (base), 00930-00500 (enhanced character set)

String Conventions:

Base Leading, imbedded, and trailing space (X'40') characters are not allowed.

ECS Leading space (X'40') characters are not allowed, trailing space characters are not significant, and imbedded space characters are significant.

Report-To_DEN

Description: The *report-to_DEN* is the second part of the user name to which distribution reports are to be sent. For FS1, this information is valid only if Dist_MU is of type TRANSPORT.

Format: Character string

CGCSGIDs: 01134-00500 (base), 00930-00500 (enhanced character set)

String Conventions:

Base Leading, imbedded, and trailing space (X'40') characters are not allowed.

ECS Leading space (X'40') characters are not allowed, trailing space characters are not significant, and imbedded space characters are significant.

Report_Service_Parms

Description:

The *report_service_parameters* structure describes the service requested for the distribution report by the origin agent when the agent wants to override the *service_parameters* that would be routinely generated by the reporting DSU for the report MU. If *report_service_parameters* are specified, they are used as the *service_parameters* in any DRMUs that are generated as part of the distribution. If the origin agent does not specify one or more of the *report_service_parameters*, a DSU that generates a report derives appropriate *service_parameters* for the DRMU from the *service_parameters* in the DTMU.

For FS2, the comparison operators and values derived for the protection and security parameters are the same as those specified (explicitly or implicitly) in the DTMU. For FS1, the comparison operators and values derived for the protection, capacity, and security parameters are the same as those specified in the Dist_MU *type* TRANSPORT.

For the priority service parameter, the value derived is either FAST or CONTROL. FAST is used if the DTMU specified FAST priority; CONTROL is used if the DTMU specified a DATA_N priority. CONTROL priority is used only in DRMUs; it may not be specified for the priority service parameter in a DTMU. If the origin agent explicitly specifies a value for the priority report service parameter, the value may be FAST, CONTROL, or DATA_N. The comparison operator for the priority service parameter is always REQUIRE_LEVEL_GE.

Format:

Special format consisting of ordered, optional *report_service_parameter* triplets of the same general structure as for *service_parameters*. See *service_parameters* on page 570.

Byte	**Content**
0-1	LT header
2-31	Up to 10 different *report_service_parameter* (RSP) triplets may be carried in one distribution. Each triplet, when present, appears in ascending sequence of parameter type. For FS2, the capacity triplet is not used in the DRMU, and therefore the capacity RSP is never specified. For FS1, the capacity triplet is used. For FS2, all service parameters are optional in both the DTMU and the DRMU. For FS1, the first three parameters—priority, protection, and capacity—are present if report service parameters are to be specified.

Priority RSP Triplet

Byte	Content		
0	X′01′		
1	X′C0′	REQUIRE_LEVEL_GE	
2	X′F0′	FAST	
	X′D0′	CONTROL	
	X′80′	DATA_16	(can be treated as DATAHI)
	X′78′	DATA_15	(can be treated as DATAHI)
	X′70′	DATA_14	(can be treated as DATAHI)
	X′68′	DATA_13	(can be treated as DATAHI)
	X′60′	DATA_12	(DATAHI)
	X′58′	DATA_11	(can be treated as DATAHI)
	X′50′	DATA_10	(can be treated as DATAHI)
	X′48′	DATA_9	(can be treated as DATAHI)
	X′40′	DATA_8	(can be treated as DATALO)
	X′38′	DATA_7	(can be treated as DATALO)
	X′30′	DATA_6	(can be treated as DATALO)
	X′28′	DATA_5	(can be treated as DATALO)
	X′20′	DATA_4	(DATALO)
	X′18′	DATA_3	(can be treated as DATALO)
	X′10′	DATA_2	(can be treated as DATALO)
	X′08′	DATA_1	(can be treated as DATALO)
	Note: All other values are reserved.		

Protection RSP Triplet

Byte	Content	
0	X′02′	
1	X′C0′	REQUIRE_LEVEL_GE
2	X′10′	LEVEL1: safe store may be performed.
	X′30′	LEVEL2: safe store must be performed.
	Note: All other values are reserved.	

Capacity RSP Triplet (not present in FS2)

Byte	Content
0	X'03'
1	X'C0' REQUIRE_LEVEL_GE
2	X'00' ZERO

Notes: All other values are reserved.
Also, All FS1 implementations are able to receive distribution reports of FOUR_K capacity (X'0C').
New FS1 implementations always send distribution reports of ZERO capacity.

Security RSP Triplet

Byte	Content
0	X'04'
1	X'C0' REQUIRE_LEVEL_GE
2	X'01' LEVEL1: security is not required. X'20' LEVEL2: security is required.

Note: All other values are reserved.

Report-To_Agent

Description: The *report-to_agent* is the name of the application transaction program to be started after the report is queued for delivery. If *report-to_agent* is absent in the DTMU, the value specified in the DTMU for *origin_agent* is used in the DRMU for *report-to_agent*.

Format: Character string, except for first byte.

CGCSGID: 01134-00500 (character set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

The first byte of an SNA-registered transaction program name ranges in value from X'00' to X'3F'. When the first byte ranges in value from X'41' to X'FF', the transaction program is not SNA registered. X'40' is not a valid first byte.

Dest_Agent

Description: The *destination_agent* is the transaction program at the destination DSU to which the distribution is to be delivered. If *dest_agent* is absent in the DTMU, the value specified for *origin_agent* is assumed to be the *dest_agent*.

Format: Character string, except for first byte

CGCSGID: 01134-00500 (character set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

The first byte of an SNA-registered transaction program name ranges in value from X'00' to X'3F'. When the first byte ranges in value from X'41' to X'FF', the transaction program is not SNA registered. X'40' is not a valid first byte.

Dest_List

Description: The *destination_list* is the list of destinations for the distribution, which can contain up to 256 destinations. Each destination is a *dest_DSU* with or without a *dest_user*, expressed as (*dest_DSU* (,*dest_user*)). For single-destination distributions and distribution reports, the *dest_list* contains only one destination.

Either a flat destination list, of the form

(*dest_DSU* (*dest_user*)), ..., (*dest_DSU* (*dest_user*)), ...

or a factored destination list, of the form

(*dest_DSU* (*dest_user*, *dest_user*, ...)), (*dest_DSU* (*dest_user*, ...))

may be present. For example, a flat destination list might contain

(DSU_A USER_1), (DSU_A USER_2), (DSU_A), (DSU_B USER_3), (DSU_B USER_4)

whereas a factored destination list would contain

(DSU_A (USER_1, USER_2)), (DSU_A), (DSU_B (USER_3, USER_4)).

Dest

Description: The *destination* associates *dest_users* with a *dest_DSU*. For flat destination lists, there are zero or one user names per *dest*. For factored destination lists, there can be multiple user names per *dest*.

Dest_DSU

Description: The *destination_DSU* is the name of one of the DSUs to which the distribution is to be sent.

Dest_RGN

Description: The *destination_RGN* is the first part of a *dest_DSU* name. This is typically, but not necessarily, the network ID.

Format: Character string

CGCSGID: 01134-00500 (character set AR)

String Conventions: Leading, imbedded, and trailing space (X′40′) characters are not allowed.

Dest_REN

Description: The *destination_REN* is the second part of a *dest_DSU* name. This is typically, but not necessarily, the LU name.

Format: Character string

CGCSGID: 01134-00500 (character set AR)

String Conventions: Leading, imbedded, and trailing space (X′40′) characters are not allowed.

Note: If a product chooses to implement DGN=REN, the enhanced character set (ECS) subset is implemented in a particular network, and any DGN contains an ECS character that is not an element of character set AR, then ECS characters may occur in this structure.

Dest_User

Description: The *destination_user* is the name of one of the users to which the distribution is to be sent.

Dest_DGN

Description: The *destination_DGN* is the first part of the name of a *dest_user*.

Format: Character string

CGCSGIDs: 01134-00500 (base), 00930-00500 (enhanced character set)

String Conventions:

Base Leading, imbedded, and trailing space (X′40′) characters are not allowed.

ECS Leading space (X′40′) characters are not allowed, trailing space characters are not significant, and imbedded space characters are significant.

Dest_DEN

Description: The *destination_DEN* is the second part of the name of a *dest_user*.

Format: Character string

CGCSGIDs: 01134-00500 (base), 00930-00500 (enhanced character set)

String Conventions:

Base: Leading, imbedded, and trailing space (X'40') characters are not allowed.

ECS: Leading space (X'40') characters are not allowed, trailing space characters are not significant, and imbedded space characters are significant.

Agent_Object

Description: The *agent_object* is directly supplied by the origin agent. It is never parsed by the distribution service and is directly delivered, unchanged, to the agent at each destination.

Format: Undefined byte string

Server_Object

Description: The *server_object* is identified by the origin agent and is fetched by the origin server when sending the *dist_transport_MU*. For FS1, the *server_object* is fetched by the origin server during transmission of the Dist_MU *type* TRANSPORT. At each destination, the *server_object* is stored by the destination server and a notification of its receipt is delivered to the destination agent.

Length Restriction: The maximum segment size for FS1 is 32511.

Format: Undefined byte string

Supplemental_Dist_Info2

Description: The *supplemental_dist_info2* structure is reserved for future use.

Format: Undefined byte string

DS_Suffix

Description: The *distribution_services_suffix* contains no information and marks the end of the *dist_transport_MU*, *dist_report_MU*, or *dist_continuation_MU*.

Dist_Report_MU

Description: The *distribution_report_message_unit* carries information reporting on the state of the distribution. Typically, for a multiple destination distribution, a *dist_report_MU* will report on only a portion of the distribution. The report is delivered to the report-to destination if one was specified in the reported-on DTMU; otherwise, it is delivered to the distribution originator.

Length Restriction: The minimum length of a *dist_report_MU* originated by an FS2 DSU is 78 bytes. This is due to the length restriction on the *Report_DTM*.

Report_Prefix

Description: The *report_prefix* identifies the beginning of *dist_report_MU*. This structure carries information that changes from DSU to DSU.

Report_Command

Description: The *report_command* contains the control information for the distribution report.

Length Restriction: The minimum length of a *dist_report_MU* originated by an FS2 DSU is 26 bytes. This is due to the length restriction on the *Report_DTM*.

Reporting_DSU

Description: The *reporting_DSU* is the name of the DSU that generated the report.

Reporting_RGN

Description: The *reporting_RGN* is the first part of the name of the DSU that generated the report. This is typically, but not necessarily, the network ID.

Format: Character string

CGCSGID: 01134-00500 (character set AR)

String Conventions: Leading, trailing, and imbedded space (X'40') characters are not allowed.

Reporting_REN

Description: The *reporting_REN* is the second part of the name of the DSU that generated the report. This is typically, but not necessarily, the LU name.

Format: Character string

CGCSGID: 01134-00500 (character set AR)

String Conventions: Leading, trailing, and imbedded space (X'40') characters are not allowed.

Report_DTM

Description:	The *report_date-time* contains the date and time at which the reporting DSU generated the report. FS2 products support the offset from GMT for local time.
Length Restriction:	Originating FS2 DSUs always generate a GMT-based time. The minimum length for *report_DTM* is therefore 11 (1-origin).
Format:	Byte string

Byte	Content
0-1	LT header
	DATE
2-3	Year, in binary (e.g., 1989 is encoded as X′07C5′)
4	Month of the year, in binary (values from 1 to 12 are valid)
5	Day of the month, in binary (values from 1 to 31 are valid)
	TIME
6	Hour of the day, in binary (values from 0 to 23 are valid)
7	Minute of the hour, in binary (values from 0 to 59 are valid)
8	Second of the minute, in binary (values from 0 to 59 are valid)
9	Hundredth of the second, in binary (values from 0 to 99 are valid)
	GMT FLAGS
10	Indicates that specified TIME is GMT and identifies whether offsets from GMT are required to calculate local time. (Equivalent EBCDIC characters are shown in parentheses.) X′E9′ (z) no offset required X′4E′ (+) add required offset to GMT to get local time X′60′ (-) subtract required offset from GMT to get local time Note: All other values are reserved.
11	Hour offset from GMT, in binary, occurs when GMT flag ≠ X′E9′ (values from 0 to 23 are valid)
12	Minute offset from GMT, in binary, occurs when GMT flag ≠ X′E9′ (values from 0 to 59 are valid)

Examples:

A 9-byte date-time encoding is a date-time followed immediately by an EBCDIC ″Z″ and is considered to be GMT. Thus, 12:00GMT on 2 January 1988 would be

```
X'07C401020C000000E9'
  yyyyMMddHHmmsshhZ
```

An 11-byte date-time encoding is a date-time followed immediately by an EBCDIC ″+″ or ″-″ and two 1-byte binary numbers, and is considered to be GMT and the offset from GMT to local time. Thus, 7:00am on 2 January 1988 in New York would be 12:00GMT - 5 hours, or

```
X'07C401020C000000600500'
  yyyyMMddHHmmsshh- HHmm
```

Report-To_DSU_User

Description: The *report-to_DSU_user* is the DSU or user to which the distribution report is being sent.

Report_Information

Description: The *report_information* identifies the distribution (or portion thereof) being reported on.

Reported-On_Origin_DSU

Description: The *reported-on_origin_DSU* is the name of the DSU at which the distribution was originated.

Presence Rules: If *reported-on_origin_DSU* is present, and *reported-on_origin_user* is absent, then the distribution was originated by a DSU; if *reported-on_origin_user* is present and *reported-on_DSU* is absent, then the report either originated in or passed through an FS1 subnetwork. If both *reported-on_origin_DSU* and *reported-on_origin_user* are present, then the report is not going to the originator of the distribution; if both *reported-on_origin_DSU* and *reported-on_origin_user* are absent, then they default to *report-to_DSU* and, if applicable, *report-to_user*.

Reported-On_Origin_RGN

Description: The *reported-on_origin_RGN* is the first part of the DSU name at which the distribution originated. This is typically, but not necessarily, the network ID.

Format: Character string

CGCSGID: 01134-00500 (character set AR)

String Conventions: Leading, trailing, and imbedded space (X′40′) characters are not allowed.

Reported-On_Origin_REN

Description: The *reported-on_origin_REN* is the second part of the DSU name at which the distribution originated. This is typically, but not necessarily, the LU name.

Format: Character string

CGCSGID: 01134-00500 (character set AR)

String Conventions: Leading, trailing, and imbedded space (X′40′) characters are not allowed.

Reported-On_Origin_User

Description: The *reported-on_origin_user* is the name of the user that originated the distribution.

Presence Rules: If *reported-on_origin_DSU* is present, and *reported-on_origin_user* is absent, then the distribution was originated by a DSU; if *reported-on_origin_user* is present and *reported-on_DSU* is absent, then the report either originated in or passed through an FS1 subnetwork. If both *reported-on_origin_DSU* and *reported-on_origin_user* are present, then the report is not going to the originator of the distribution; if both *reported-on_origin_DSU* and *reported-on_origin_user* are absent, then they default to *report-to_DSU* and, if applicable, *report-to_user*.

Reported-On_Origin_DGN

Description: The *reported-on_origin_DGN* is the first part of the name of the user that originated the distribution.

Format: Character string

CGCSGIDs: 01134-00500 (base), 00930-00500 (enhanced char set)

String Conventions:

Base Leading, trailing, and imbedded space (X'40') characters are not allowed.

ECS Leading space (X'40') characters are disallowed, trailing space characters are not significant, and imbedded space characters are significant.

Reported-On_Origin_DEN

Description: The *reported-on_origin_DEN* is the second part of the name of the user that originated the distribution.

Format: Character string

CGCSGIDs: 01134-00500 (base), 00930-00500 (enhanced char set)

String Conventions:

Base Leading, trailing, and imbedded space (X'40') characters are not allowed.

ECS Leading space (X'40') characters are disallowed, trailing space characters are not significant, and imbedded space characters are significant.

Reported-On_Seqno_DTM

Description:	The *reported-on_sequence_number/date-time*, in combination with the origin agent, origin DSU, and origin user, is the unique identifier of the distribution. The origin agent, origin DSU, and origin user are specified in the appropriate reported-on or report-to structures. The sequence number is the number assigned to the distribution by the origin agent. For FS2, the number ranges from 1 to (2**31)-1. For FS1, the number ranges from 1 to 9999. Refer to Appendix D for migration details. The date-time is the date and time generated at the origin of the distribution. FS2 products support the offset from GMT for local time.
Length Restriction:	Originating FS2 DSUs always generate a GMT-based time. The minimum length for *reported-on_seqno_DTM* is 15 (1-origin).
Format:	Byte string

Byte	Content
0-1	LT header
	SEQNO
2-5	Signed binary integer limited to (2**31)-1
	DATE
6-7	Year, in binary (e.g., 1989 is encoded as X'07C5')
8	Month of the year, in binary (values from 1 to 12 are valid)
9	Day of the month, in binary (values from 1 to 31 are valid)
	TIME
10	Hour of the day, in binary (values from 0 to 23 are valid)
11	Minute of the hour, in binary (values from 0 to 59 are valid)
12	Second of the minute, in binary (values from 0 to 59 are valid)
13	Hundredth of the second, in binary (values from 0 to 99 are valid)
	GMT FLAGS
14	Indicates that specified TIME is GMT and identifies whether offsets from GMT are required to calculate local time. (Equivalent EBCDIC characters are shown in parentheses.) X'E9' (z) no offset required X'4E' (+) add required offset to GMT to get local time X'60' (-) subtract required offset from GMT to get local time Note: All other values are reserved.
15	Hour offset from GMT, in binary, occurs when GMT flag ≠ X'E9' (values from 0 to 23 are valid)
16	Minute offset from GMT, in binary, occurs when GMT flag ≠ X'E9' (values from 0 to 59 are valid)

Examples:

A 9-byte date-time encoding is a date-time followed immediately by an EBCDIC "Z" and is considered to be GMT. Thus, 12:00GMT on 2 January 1988 would be

```
X'07C401020C000000E9'
  yyyyMMddHHmmsshhZ
```

An 11-byte date-time encoding is a date-time followed immediately by an EBCDIC "+" or "-" and two 1-byte binary numbers, and is considered to be GMT and the offset from GMT to local time. Thus, 7:00am on 2 January 1988 in New York would be 12:00GMT - 5 hours, or

```
X'07C401020C000000600500'
  yyyyMMddHHmmsshh- HHmm
```

Reported-On_Supp_Dist_Info1

Description: The *reported-on_supp_dist_info1* structure is reserved for future use.

Format: Character string

Reported-On_Agent_Correl

Description: The *reported-on_agent_correlation* is a string that was supplied by the origin agent at the origin DSU.

Format: Undefined byte string

Reported-On_Origin_Agent

Description: The *reported-on_origin_agent* is the name of the transaction program at the origin DSU that originated the distribution that is being reported on.

Presence Rule: Occurs when *report-to_agent* is different from *origin_agent*. If third-party reporting has been requested and a report was generated in or flowed through an FS1 subnetwork, the *reported-on_origin_agent* structure is discarded.

Format: Character string, except for first byte

CGCSGID: 01134-00500 (character set AR)

String Conventions: Leading, trailing, and imbedded space (X'40') characters are not allowed.

The first byte of an SNA-registered transaction program name ranges in value from X'00' to X'3F'. When the first byte ranges in value from X'41' to X'FF', the transaction program is not SNA registered. X'40' is not a valid first byte.

Reported-On_Dest_Agent

Description: The *reported-on_destination_agent* is the name of the transaction program at the destination DSU that was specified for the reported-on distribution.

Presence Rule: Occurs when *dest_agent* was specified in the reported-on DTMU.

Format: Character string, except for first byte

CGCSGID: 01134-00500 (character set AR)

String Conventions: Leading, trailing, and imbedded space (X'40') characters are not allowed.

The first byte of an SNA-registered transaction program name ranges in value from X'00' to X'3F'. When the first byte ranges in value from X'41' to X'FF', the transaction program is not SNA registered. X'40' is not a valid first byte.

Reported-On_Supp_Dist_Info2

Description: The *reported-on_supp_dist_info2* structure is reserved for future use.

Format: Undefined byte string

Dist_Continuation_MU

Description: The *distribution_continuation_message_unit* is used by a sending DSU to continue transmission of a suspended MU.

Continuation_Prefix

Description: The *continuation_prefix* identifies the beginning of a DCMU.

Restarting_Byte_Position

Description: The *restarting_byte_position* indicates where the sender is beginning retransmission of the first structure being re-sent. The byte count begins with the first byte of atomic data (i.e., no LLs included) within the encompassing structure. Absence of this structure is equivalent to the presence of a 1 in this structure, implying that the first structure present in the DCMU is being re-sent in its entirety. 0 is not allowed.

Format: Unsigned binary integer (1-origin)

SNA_Condition_Report

Description: The *SNA_condition_report* describes the condition being reported. The condition is always identified by an *SNA_report_code.*

Certain conditions can be more fully described by supplementary information. Conditions pertaining to one or more structures in a format can have the location and contents of each of those structures specified by a *structure_report*. Certain conditions arise from inconsistencies among multiple portions of the MU. Each portion is described by a separate *structure_report*.

SNA_Report_Code

Description: The *SNA_report_code* is an SNA registered code identifying the condition that is being reported. Refer to Appendix E for allowable values and descriptions.

Format: Byte string

Byte	Content
0-1	LT header
2-3	Primary report code
4-5	Subcode

Structure_Report

Description: The *structure_report* reports on a structure involved in a format-related condition. Depending on the condition, the *structure_report* may describe a structure that was present in, or absent from, the reported-on MU.

A format condition has its location in the MU pinpointed by a *structure_spec* and a list of *parent_spec*s that define a line-of-descent. The line-of-descent begins with the MU and continues down the parent-child hierarchy to a level as low as the particular condition warrants. A registered ID always appears in a *structure_report*; if the reported-on structure is not itself a registered ID, its line-of-descent is traced up to include a registered ancestor.

Presence Rule: Presence governed by the *SNA_report_code*.

Structure_State

Description: The *structure_state* indicates whether the reported-on structure was present or absent.

Format: Hexadecimal code

Byte	Content
0-1	LT header
2	X'01' STRUCTURE_PRESENT X'02' STRUCTURE_ABSENT Note: All other values are reserved.

Structure_Contents

Description: The *structure_contents* is the portion of the MU that is relevant to the detected condition. Typically, the *structure_contents* contains the header of the structure and at least the beginning of its contents. When the condition can be isolated to a portion of the structure, the *structure_contents* contains only that portion of the structure relevant to the condition. In this case, the *structure_segment_number* and *structure_byte_offset* locate the portion of the structure relevant to the condition.

Presence Rule: Allowed only when *structure_state* = STRUCTURE_PRESENT.

Format: Undefined byte string

Parent_Spec

Description: The *parent_specification* contains the identifier (ID or T) and the class of a parent structure. For a parent structure that occurs multiple times, the instance may also be included. The value of the *parent_instance* identifies the particular instance. The position of this parent structure within its parent (if one exists) may also be included. This would typically be done when this parent structure is an unordered child of its parent.

Parent_ID_Or_T

Description: The *parent_ID_or_T* is the ID or T value of a parent structure. ID values are the registered GDS codepoints. T values are architecture-specific values relative to the encompassing ID.

Format: Undefined byte string

Parent_Class

Description: The *parent_class* is the class of a parent structure.

Presence Rule: If absent, defaults to LENGTH-BOUNDED_LT_STRUCTURE.

Format: Hexadecimal code

Byte	Content	
0-1	LT header	
2	X'01'	LENGTH-BOUNDED_LLID_STRUCTURE (ID)
	X'02'	LENGTH-BOUNDED_LT_STRUCTURE (T) (default)
	X'03'	DELIMITED_LLID_STRUCTURE (DEL-ID)
	X'04'	DELIMITED_LT_STRUCTURE (DEL-T)
	X'05'	IMPLIED_LLID_STRUCTURE (IMP-ID)
	X'06'	IMPLIED_LT_STRUCTURE (IMP-T)
	Note: All other values are reserved.	

Parent_Position

Description: The *parent_position* is the position of this parent structure within its parent (if one exists) in this particular MU. Multiple consecutive instances of a repeatable parent structure share a single position, and can be distinguished by *parent_instance*.

Format: Signed binary integer

Parent_Instance

Description: The *parent_instance* is used when a parent structure occurs multiple times. The value of *parent_instance* identifies the particular instance within a position.

Format: Signed binary integer

Structure_Spec

Description: The *structure_specification* contains the identifier (ID or T) and the class of a structure. For a structure that occurs multiple times, the instance may also be included. The value of the *structure_instance* identifies the particular instance. The position of this structure within its parent structure may also be included. This would typically be done when the parent structure contains unordered children.

Presence Rule: Absent only when the *structure_class* is the default and the *sibling_list* contains all pertinent ID or T values.

Structure_ID_Or_T

Description: The *structure_ID_or_T* is the ID or T value of the structure. ID values are the registered GDS codepoints. T values are architecture-specific values relative to the encompassing ID.

Presence Rule: Required except when *sibling_list* contains all pertinent ID or T values. In this case, the structures specified by *sibling_list* are the structures being reported on.

Format: Undefined byte string

Structure_Class

Description: The *structure_class* is the class of the reported-on structure and any siblings identified in *sibling_list*.

Presence Rule: If absent, defaults to LENGTH-BOUNDED_LT_STRUCTURE.

Format: Hexadecimal code

Byte	Content	
0-1	LT header	
2	X'01'	LENGTH-BOUNDED_LLID_STRUCTURE (ID)
	X'02'	LENGTH-BOUNDED_LT_STRUCTURE (T) (default)
	X'03'	DELIMITED_LLID_STRUCTURE (DEL-ID)
	X'04'	DELIMITED_LT_STRUCTURE (DEL-T)
	X'05'	IMPLIED_LLID_STRUCTURE (IMP-ID)
	X'06'	IMPLIED_LT_STRUCTURE (IMP-T)
	Note: All other values are reserved.	

Structure_Position

Description: The *structure_position* is either the actual or expected position of this structure within its parent in this particular MU. Multiple consecutive instances of a repeatable structure share a single position, and can be distinguished by *structure_instance*.

Format: Signed binary integer (1-origin)

Structure_Instance

Description: The *structure_instance* is used when the structure is one of multiple occurrences of a repeatable structure. The value of *structure_instance* identifies the particular instance within a position.

Format: Signed binary integer (1-origin)

Structure_Segment_Number

Description: The *structure_segment_number* is the segment of the structure in which the condition was detected.

Presence Rule: Occurs when the beginning of *structure_contents* was not contained in the first segment of the reported-on structure.

Format: Signed binary integer (1-origin)

Structure_Byte_Offset

Description: The *structure_byte_offset* marks the start of *structure_contents* within the reported-on structure. If *structure_segment_number* is present, this value is the offset from the start of the indicated segment; otherwise, it is the offset from the beginning of the structure.

Format: Signed binary integer (0-origin)

Sibling_List

Description: The *sibling_list* contains a string of ID or T values necessary to describe the detected condition. The structures identified in *sibling_list* are children of the parent identified in *parent_spec* and/or siblings of the structure identified in *structure_spec*. The class of the sibling structures is the same as *structure_class*. The expected position, when applicable, is given by *structure_position*.

Presence Rule: Presence is governed by the *SNA_report_code*.

Format: Byte string

Reported-On_Dest_List

Description: The *reported-on_destination_list* contains the portion of the distribution destinations that are being reported on.

Presence Rule: Presence is governed by the *SNA_report_code*.

Reported-On_Dest_Prefix

Description: The *reported-on_destination_prefix* is the prefix of the *reported-on_destination_list*.

Reported-On_Dest

Description: The *reported-on_destination* associates *reported-on_dest_user*s with a *reported-on_dest_DSU* for those destinations specified in the original distribution request being reported on. For flat destination lists (i.e., lists containing only DSUs and/or DSU-user pairs), there are zero or one user names per DSU list. For factored destination lists, there can be multiple user names per DSU list.

Reported-On_Dest_DSU

Description: The *reported-on_destination_DSU* is one of the original destination DSUs being reported on.

Reported-On_Dest_RGN

Description: The *reported-on_destination_RGN* is the first part of the name of one of the original destination DSUs being reported on. This is typically, but not necessarily, the network ID.

Presence Rule: Absent when passed through an FS1 subnetwork.

Format: Character string

CGCSGID: 01134-00500 (character set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

Reported-On_Dest_REN

Description: The *reported-on_destination_REN* is the second part of the name of one of the original destination DSUs being reported on. This is typically, but not necessarily, the LU name.

Presence Rule: Absent when passed through an FS1 subnetwork.

Format: Character string

CGCSGID: 01134-00500 (character set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

Note: If a product chooses to implement DGN = REN, the ECS subset is implemented in a particular network, and any DGN contains an ECS character that is not an element of Character Set AR, then ECS characters may occur in this structure.

Reported-On_Dest_User

Description: The *reported-on_destination_user* is the name of one of the original destination users being reported on.

Reported-On_Dest_DGN

Description: The *reported-on_destination_DGN* is the first part of the name of one of the original destination users being reported on.

Note: In FS1, for a DS condition code of X'000D' (lost user names), user names will be null.

Format: Character string

CGCSGID: 01134-00500 (base), 00930-00500 (enhanced character set)

String Conventions:

Base Leading, imbedded, and trailing space (X'40') characters are not allowed.

ECS Leading space (X'40') characters are not allowed, trailing space characters are not significant, and imbedded space characters are significant.

Reported-On_Dest_DEN

Description: The *reported-on_destination_DEN* is the second part of the name of one of the original destination users being reported on.

Note: In FS1, for a DS condition code of X'000D' (lost user names), user names will be null.

Format: Character string

CGCSGID: 01134-00500 (base), 00930-00500 (enhanced character set)

String Conventions:

Base Leading, imbedded, and trailing space (X'40') characters are not allowed.

ECS Leading space (X'40') characters are not allowed, trailing space characters are not significant, and imbedded space characters are significant.

Reported-On_Dest_Suffix

Description: The *reported-on_destination_suffix* is the suffix of the *reported-on_destination_list*.

Supplemental_Report

Description: The *supplemental_report* contains other information pertaining to a condition. The contents of the *supplemental_report* are governed by the *SNA_report_code.*

Presence Rule: Presence is governed by the *SNA_report_code*.

Sender_Exception_MU

Description: The *sender_exception_MU* is sent from the sender to the receiver when the sender detects an exception while sending a *dist_transport_MU*, a *dist_report_MU*, or a *dist_continuation_MU*.

Receiver_Exception_MU

Description: The *receiver_exception_MU* is sent from the receiver to the sender when the receiver detects an exception while receiving a *dist_transport_MU*, a *dist_report_MU*, or a *dist_continuation_MU*.

Receiver_Exception_Command

Description: The *receiver_exception_command* is the prefix identifying the *receiver_exception_MU*.

Sender_Retry_Action

Description: The *sender_retry_action* is the receiver's recommendation to the sender as to whether to retry the transmission of the MU.

Format: Hexadecimal code

Byte	Content	
0-1	LT header	
2	X'01'	RETRY_PRECLUDED
	X'02'	RETRY_ALLOWED
	X'03'	RETRY_EXPECTED_USING_DCMU
	Note: All other values are reserved.	

Receiving_DSU

Description: The *receiving_DSU* is the name of the DSU to which a distribution was being sent.

Receiving_RGN

Description: The *receiving_RGN* is the first part of the name of the DSU to which a distribution was being sent. This is typically, but not necessarily, the network ID.

Format: Character string

CGCSGID: 01134-00500 (character set AR)

String Conventions: Leading, imbedded, and trailing space (X'40') characters are not allowed.

Receiving_REN

Description: The *receiving_REN* is the second part of the name of the DSU to which a distribution was being sent. This is typically, but not necessarily, the LU name.

Format: Character string

CGCSGID: 01134-00500 (character set AR)

String Conventions: Leading, imbedded, and trailing space (X′40′) characters are not allowed.

Note: If a product chooses to implement DGN=REN, the enhanced character set (ECS) subset is implemented in a particular network, and any DGN contains an ECS character that is not an element of SNA Character Set AR, then ECS characters may occur in this structure.

Completion_Query_MU

Description: The *completion_query_message_unit* is sent by the sending DSU to query the completion status of a particular MU at the receiving DSU.

Completion_Report_MU

Description: The *completion_report_message_unit* is sent by the receiving DSU to report on the completion status of a particular MU or to control traffic flow on a conversation.

Indicator_Flags

Description: The *indicator_flags* structure contains a 1-byte flag, to indicate the completion status of the *MU_ID* identified in a *completion_report_MU*, or to control traffic flow on a conversation.

Format: Bit string

Note: Conversation control flags (bits 2 and 3) may be used in conjunction with flow control flags (Not Received, In Transit, Suspended, Terminated, Completed, Purged).

Bit Map	**Architecturally-Defined Value**
0 1 2 3 4 5 6 7	
x x 0 0 x x x x	Default—Normal DS flow
x x 0 1 x x x x	Terminate Conversation
0 x x x 0 0 0 0	Not Received
0 x x x 0 0 0 1	In Transit
0 x x x 0 0 1 0	Suspended
0 x x x 0 0 1 1	Completed
0 x x x 0 1 0 1	Terminated
1 x x x x x x x	Purged
Note: x = any value	

Last_Structure_Received

Description: *Last_structure_received* is the codepoint of the structure the receiving DSU identifies as the last structure received before the MU was suspended. This structure must be a length-bounded LLID structure at the highest level of the MU.

Presence Rule: If *indicator_flags* = SUSPENDED, then *last_structure_received* is present.

Format: Hexadecimal code

Last_Byte_Received

Description: *Last_byte_received* is the last byte received by the receiving DSU before the MU was suspended. The byte count begins with the first byte of atomic data within the encompassing structure. A byte count of X′FFFFFFFFFFFFFFFF′ indicates that the structure was fully received. The byte count contains only atomic data and does not contain the segmenting LLs for segmented structures.

Presence Rules: If *indicator_flags* = SUSPENDED, *last_structure_received* is present, and *last_byte_received* is absent, then the structure was received.

Format: Unsigned binary integer (1-origin)

Purge_Report_MU

Description: The *purge_report_message_unit* indicates to the receiving DSU that the sending DSU has marked a particular *MU_ID* PURGED, and that the receiving DSU may flag that *MU_ID* as PURGED.

Reset_Request_MU

Description: The *reset_request_message_unit* is sent from DS_Send to DS_Receive. DS_Send issues the *reset_request_MU* to request that DS_Receive reset its *MU_ID* registry.

Reset_DTM

Description: The *reset_date-time* contains the date and time at which the *reset_request_MU* was generated. Both sender and receiver store it as the "time of last reset" of their *MU_ID* registries.

Length Restriction: Originating FS2 DSUs always generates a GMT-based time. The minimum length for *reset_DTM* is 11 (1-origin).

Format: Byte string

Byte	Content
0-1	LT header
	DATE
2-3	Year, in binary (e.g., 1989 is encoded as X'07C5')
4	Month of the year, in binary (values from 1 to 12 are valid)
5	Day of the month, in binary (values from 1 to 31 are valid)
	TIME
6	Hour of the day, in binary (values from 0 to 23 are valid)
7	Minute of the hour, in binary (values from 0 to 59 are valid)
8	Second of the minute, in binary (values from 0 to 59 are valid)
9	Hundredth of the second, in binary (values from 0 to 99 are valid)
	GMT FLAGS
10	Indicates that specified TIME is GMT and identifies whether offsets from GMT are required to calculate local time. (Equivalent EBCDIC characters are shown in parentheses.) X'E9' (z) no offset required X'4E' (+) add required offset to GMT to get local time X'60' (-) subtract required offset from GMT to get local time Note: All other values are reserved.
11	Hour offset from GMT, in binary, occurs when GMT flag ≠ X'E9' (values from 0 to 23 are valid)
12	Minute offset from GMT, in binary, occurs when GMT flag ≠ X'E9' (values from 0 to 59 are valid)

Examples:

A 9-byte date-time encoding is a date-time followed immediately by an EBCDIC "Z" and is considered to be GMT. Thus, 12:00GMT on 2 January 1988 would be

```
X'07C401020C000000E9'
  yyyyMMddHHmmsshhZ
```

An 11-byte date-time encoding is a date-time followed immediately by an EBCDIC "+" or "-" and two 1-byte binary numbers, and is considered to be GMT and the offset from GMT to local time. Thus, 7:00am on 2 January 1988 in New York would be 12:00GMT - 5 hours, or

```
X'07C401020C000000600500'
  yyyyMMddHHmmsshh- HHmm
```

Reset_Accepted_MU

Description: The *reset_accepted_message_unit* is sent from DS_Receive to DS_Send. DS_Receive issues the *reset_accepted_MU* in response to a *reset_request_MU* to inform DS_Send that DS_Receive has reset its MU_ID Registry.

Unrecognized_Reserve

Description: The *unrecognized_reserve* is the number of bytes reserved for unrecognized structures. An unrecognized structure occurs within its parent structure. The number of unrecognized structures allowable for a particular parent structure is limited by the number of children allowable for that parent structure.

Intermediate FS2 DSUs pass *unrecognized_reserve* structures through unchanged in outgoing DMUs.

Format: Undefined byte string

Header Description Tables for FS1 Message Units

DISTRIBUTION MESSAGE UNIT (DIST_MU)

Table 80 (Page 1 of 2). Distribution Message Unit (DIST_MU)

Structure Name	Struct Ref Pg	Struct Class	IDF/T	Length	Occur-rences	Children			
						Unrec	Order	Num	Sub Table
Dist_MU	610	Del-IDF	pfx	≥148	1	N	Y	3-4	—
Prefix	610	IDF/pfx	C00102	5-21	1	—	—	—	—
Dist_Command	610	IDF/seg	C10502	138-32511	1	N	Y	2-3	—
Service_Desc_Operands	610	Imp-IDF	idc	58-774	1	N	N	2-5	—
Dist_ID	610	IDF/idc	C34041	28-107	1	N	N	5-7	—
Origin_RGN	574	T	01	3-10	0-1	—	—	—	—
Origin_REN	574	T	02	3-10	1	—	—	—	—
Origin_DGN	574	T	03	2-10	1	—	—	—	—
Origin_DEN	575	T	04	2-10	1	—	—	—	—
Origin_Seqno	611	T	05	6	1	—	—	—	—
Origin_DTM	611	T	06	10	1	—	—	—	—
Agent_Correl	577	T	07	3-46	0-1	—	—	—	—
Dist_Gen_Options	611	IDF	C33D41	30-58	1	N	N	5	—
Dist_Flags (FS1)	612	T	01	3	1	—	—	—	—
Hop_Count	568	T	02	4	1	—	—	—	—
Service_Parms	570	T	03	11-32	1	—	—	—	—
Server_Object_Ind	612	T	04	4	1	—	—	—	—
Origin_Agent	573	T	05	3-10	1	—	—	—	—
Report-To_Address	612	IDF	C36041	14-45	0-1*	N	N	3-4	—
Report-To_RGN	577	T	01	3-10	0-1	—	—	—	—
Report-To_REN	577	T	02	3-10	1	—	—	—	—
Report-To_DGN	578	T	03	3-10	1	—	—	—	—
Report-To_DEN	578	T	04	3-10	1	—	—	—	—
Report-To_Options	613	IDF	C34341	8-47	0-1*	N	N	1-2	—
Report_Service_Parms	579	T	01	11-32	0-1	—	—	—	—
Report-To_Agent	581	T	02	3-10	0-1	—	—	—	—
Agent_Object	584	IDF	C32D01	6-517	0-1	—	—	—	—
Destination_Operands	613	Imp-IDF	idc	≥75	1	N	Y	3	—
Begin_Dest_Operands	614	IDF/idc	C35001	8	1	—	—	—	—
Dest_RGN_List	614	Imp-IDF	idc	≥62	≥1	N	Y	4	—
Dest_RGN	583	IDF/idc	C35201	5-13	1	—	—	—	—
Begin_REN_List	614	IDF	C35001	8	1	—	—	—	—
Dest_REN_List	614	Imp-IDF	idc	≥44	≥1	N	Y	4	—
Dest_REN	583	IDF/idc	C35301	6-13	1	—	—	—	—
Begin_DGN_List	615	IDF	C35001	8	1	—	—	—	—
Dest_DGN_List	615	Del-IDF	pfx	≥25	≥1	N	Y	4	—

Table 80 (Page 2 of 2). Distribution Message Unit (DIST_MU)

Structure Name	Struct Ref Pg	Struct Class	IDF/T	Length	Occur-rences	Children			
						Unrec	Order	Num	Sub Table
Dest_DGN	583	IDF/pfx	C35401	6-13	1	—	—	—	—
Begin_DEN_List	615	IDF	C35001	8	1	—	—	—	—
Dest_DEN	584	IDF	C35501	6-13	≥1	—	—	—	—
End_DEN_List	615	IDF/sfx	C35101	5	1	—	—	—	—
End_DGN_List	615	IDF	C35101	5	1	—	—	—	—
End_REN_List	615	IDF	C35101	5	1	—	—	—	—
End_Dest_Operands	615	IDF	C35101	5	1	—	—	—	—
Dist_Report_Operands	616	Imp-IDF	idc	≥63	0-1*	N	Y	2-4	608
Dist_Server_Operands	615	Imp-IDF	idc	≥14	0-1*	N	Y	2	—
Server_Prefix	615	IDF/idc	C90A41	8-280	1	N	N	1-3	—
Server_Obj_Byte_Count	573	T	01	10	0-1	—	—	—	—
Server	573	T	02	3-10	1	—	—	—	—
Server_Parms	616	T	03	3-255	0-1	—	—	—	—
Server_Object	584	IDF/seg	C90801	≥6*	1	—	—	—	—
DS_Suffix (FS1)	616	IDF	CF0100	5	1	—	—	—	—

Note:

- * Refer to FS1 Structure Descriptions starting on page 610 for presence rules and length restrictions.
- *Dist_Report_Operands* does not occur for Dist_MU *type* TRANSPORT.
- *Agent_Correl*, *Report-To_Address*, *Report-To_Options*, *Agent_Object*, and *Dist_Server_Operands* do not occur for Dist_MU *type* REPORT.
- *Dest_RGN_List*, *Dest_REN_List*, *Dest_DGN_List*, and *Dest_DEN* occur only one time for Dist_MU *type* REPORT.

DIST REPORT OPERANDS

Table 81. Distribution Report Operands

Structure Name	Struct Ref Pg	Struct Class	IDF/T	Length	Occur-rences	Children			
						Unrec	Order	Num	Sub Table
Dist_Report_Operands	616	Imp-IDF	idc	≥63	0-1	N	Y	2-4	—
Report_Operands	616	Imp-IDF	idc	27-112	1	N	N	1-2	—
Report_Correlation	616	IDF/idc	C34041	27-87	1	N	N	4-5	—
Reported-On_Origin_DGN	589	T	03	3-10	1	—	—	—	—
Reported-On_Origin_DEN	589	T	04	3-10	1	—	—	—	—
Reported-On_Seqno	616	T	05	6	1	—	—	—	—
Reported-On_DTM	617	T	06	10	1	—	—	—	—
Reported-On_Agent_Correl	592	T	07	3-46	0-1	—	—	—	—
Receiving_DSU	601	IDF	C36141	8-25	0-1	N	N	1-2	—
Receiving_RGN	601	T	01	3-10	0-1	—	—	—	—
Receiving_REN	602	T	02	3-10	1	—	—	—	—
Gen_SNADS_Report	617	Imp-IDF	idc	16	0-1*	N	Y	2	—
Gen_SNADS_Type	617	IDF/idc	C35601	7	1	—	—	—	—
Gen_SNADS_Contents	618	IDF	C35741	9	1	N	Y	1	—
Gen_SNADS_Cond_Code	618	T	01	4	1	—	—	—	—
Gen_DIA_Report	618	Imp-IDF	idc	14-524	0-1*	N	Y	2	—
Gen_DIA_Type	619	IDF/idc	C35601	7	1	—	—	—	—
Gen_DIA_Contents	619	IDF	C35741	7-517*	1	—	—	—	—
Specific_Report	619	Imp-IDF	idc	≥36	1	N	Y	3	—
Begin_Report_DGN_List	619	IDF/idc	C35001	8	1	—	—	—	—
Report_DGN_List	619	Imp-IDF	idc	≥23	≥1	N	Y	4	—
Reported-On_Dest_DGN	600	IDF/idc	C35401	5-13	1	—	—	—	—
Begin_Report_DEN_List	619	IDF	C35001	8	1	—	—	—	—
Report_DEN_List	620	Imp-IDF	idc	5-553	≥1	N	Y	1-3	—
Reported-On_Dest_DEN	600	IDF/idc	C35501	5-13	1	—	—	—	—
Spec_SNADS_Report	620	Imp-IDF	idc	16	0-1*	N	Y	2	—
Spec_SNADS_Type	620	IDF/idc	C35601	7	1	—	—	—	—
Spec_SNADS_Cont	620	IDF	C35741	9	1	N	Y	1	—
Spec_SNADS_CC	621	T	01	4	1	—	—	—	—
Spec_DIA_Report	621	Imp-IDF	idc	14-524	0-1*	N	Y	2	—
Spec_DIA_Type	622	IDF/idc	C35601	7	1	—	—	—	—
Spec_DIA_Contents	622	IDF	C35741	7-517*	1	—	—	—	—
End_Report_DEN_List	622	IDF	C35101	5	1	—	—	—	—
End_Report_DGN_List	622	IDF	C35101	5	1	—	—	—	—

Note: * Refer to FS1 Structure Descriptions starting on page 610 for presence rules and length restrictions.

SENDER EXCEPTION MESSAGE UNIT (TYPE FS1)

Table 82. Sender Exception Message Unit (type FS1)

Structure Name	Struct Ref Pg	Struct Class	IDF/T	Length	Occur-rences	Children			
						Unrec	Order	Num	Sub Table
Sender_Exception_MU (FS1)	623	IDF	CF0201	8	1	—	—	—	—

RECEIVER EXCEPTION MESSAGE UNIT (TYPE FS1)

Table 83. Receiver Exception Message Unit (type FS1)

Structure Name	Struct Ref Pg	Struct Class	IDF/T	Length	Occur-rences	Children			
						Unrec	Order	Num	Sub Table
Receiver_Exception_MU (FS1)	601	Del-IDF	pfx	59-863	1	N	Y	3	—
Prefix	610	IDF/pfx	C00102	5	1	—	—	—	—
Receiver_Exception_Command	623	IDF	C10101	49-853	1	N	Y	2	—
Receiver_Exception_Correl	624	IDF	C32801	7-23	1	—	—	—	—
Exception_And_Reply_Data	624	Imp-IDF	idc	37-825	1	N	N	2	—
Receiver_Exception_Code	625	IDF/idc	C32201	8-255	1	—	—	—	—
Reply_Data	626	IDF	C34501	29-570	1	N	Y	2-3	—
Receiving_DSU	601	IDF	C36141	8-25	1	N	N	1-2	—
Receiving_RGN	601	T	01	3-10	0-1	—	—	—	—
Receiving_REN	602	T	02	3-10	1	—	—	—	—
SNADS_Report	626	Imp-IDF	idc	16	1	N	Y	2	—
SNADS_Report_Type	626	IDF/idc	C35601	7	1	—	—	—	—
SNADS_Report_Cont	626	IDF	C35741	9	1	N	Y	1	—
SNADS_Report_CC	627	T	01	4	1	—	—	—	—
DIA_Report	627	Imp-IDF	idc	14-524	0-1	N	Y	2	—
DIA_Report_Type	627	IDF/idc	C35601	7	1	—	—	—	—
DIA_Report_Cont	628	IDF	C35741	7-517	1	—	—	—	—
DS_Suffix (FS1)	616	IDF/sfx	CF0100	5	1	—	—	—	—

FS1 Structure Descriptions

Dist_MU

Description: The *distribution_message_unit* transports user information to one or more distribution service users. A Dist_MU can be one of two types based on the value of *dist_flags* (type FS1): TRANSPORT or REPORT. A Dist_MU *type* TRANSPORT transports agent and/or server objects. A Dist_MU *type* REPORT transports information reporting on the state of the distribution.

Prefix

Description: The *prefix* identifies the beginning of a message unit and may contain a message-unit identifier.

Format: Undefined byte string

Dist_Command

Description: The *distribution_command* contains all information used by each DSU to transport the distribution for a Dist_MU *type* TRANSPORT. For a Dist_MU *type* REPORT, the *distribution_command* contains the control information for the distribution report.

Service_Desc_Operands

Description: The *service_description_operands* contain all the information, except for the destination list, required by each DSU to transport the distribution.

Dist_ID

Description: The *distribution_identifier* contains information corresponding to the distribution originator.

Origin_Seqno

Description: The *origin_sequence_number* is the number assigned to the distribution by the *origin_DSU*. The value ranges from 1 to 9999 for a Dist_MU *type* TRANSPORT, and is always 0 for a Dist_MU *type* REPORT.

Format: Character string; each character is the EBCDIC representation of one digit of the sequence number.

Byte	**Content**
0-1	LT header
2-5	Sequence number

Notes:

- For Dist_MU *type* TRANSPORT, values range from X'F0F0F0F1' to X'F9F9F9F9'.
- For Dist_MU *type* REPORT, value is X'F0F0F0F0'.

Origin_DTM

Description: The *origin_date-time* is the date and time the distribution was originated by the origin DSU. Time is assumed to be local.

Format: Byte string

Byte	**Content**
0-1	LT header
	DATE
2-3	Year, in binary (e.g., 1989 is encoded as X'07C5')
4	Month of the year, in binary (values from 1 to 12 are valid)
5	Day of the month, in binary (values from 1 to 31 are valid)
	TIME
6	Hour of the day, in binary (values from 0 to 23 are valid)
7	Minute of the hour, in binary (values from 0 to 59 are valid)
8	Second of the minute, in binary (values from 0 to 59 are valid)
9	Hundredth of the second, in binary (values from 0 to 99 are valid)

Example:

The date-time encoding for 12:00 noon on 2 January 1988 is:

```
X'07C401020C000000'
  yyyyMMddHHmmsshh
```

Dist_Gen_Options

Description: The *distribution_general_options* contains structures used by DS to condition its processing of the distribution.

Dist_Flags (type FS1)

Description: The *distribution_flags* indicate reporting services requested by the origin agent.

Format:

Bit	Content
0	Exception Report bit: 0 DS is requested to generate a report in case of an exception. 1 A report will not be generated by DS for this distribution.
1	Distribution Message Unit type bit: 0 Distribution is of type TRANSPORT. 1 Distribution is of type REPORT.
2-7	Reserved

Byte	Content
0-1	LT header
2	X'00' Dist_MU *type* TRANSPORT with report requested X'80' Dist_MU *type* TRANSPORT with no report requested X'C0' Dist_MU *type* REPORT with no report requested Note: All other values are reserved.

Server_Object_Ind

Description: The *server_object_indicator* indicates whether a *server_object* is present or not. The only values supported are 0 and 1.

Presence Rule: Contains X'0001' only for Dist_MU *type* TRANSPORT.

Format: Hexadecimal code

Byte	Content
0-1	LT header
2-3	X'0000' no *server_object* present in this MU X'0001' a *server_object* present in this MU Note: All other values are reserved.

Report-To_Address

Description: The *report-to_address* contains the name of the DSU and user to which any distribution reports are sent.

Presence Rule: This information may be present only in Dist_MU *type* TRANSPORT.

Report-To_Options

Description: The *report-to_options* contains information involved in processing any reports generated as part of the distribution.

Presence Rule: This information may be present only in Dist_MU *type* TRANSPORT.

Destination_Operands

Description: The *destination_operands* are the list of destinations for the distribution. Up to 256 destinations are allowed if the distribution is of type TRANSPORT; exactly one destination, if the distribution is of type REPORT. The destinations are encoded as a fully factored, partially factored, or unfactored list of users and DSUs (see the following example).

Example: The following is a list of destinations (qualified by RGN.REN.DGN.DEN):
A.K.DA.U1, A.K.DA.U2, A.K.DB.U3, A.K.DB.U4,
A.L.DC.U5, A.L.DC.U6, A.L.DD.U7, A.L.DD.U8,
B.M.DE.U9, B.M.DE.U10, B.M.DF.U11, B.M.DF.U12,
B.N.DG.U13, B.N.DG.U14, B.N.DH.U15, and B.N.DH.U16.
The list may appear factored in *destination_operands* as follows:

- Fully factored:

```
A(K(DA(U1
       U2)
    DB(U3
       U4))
  L(DC(U5
       U6)
    DD(U7
       U8)))
B(M(DE(U9
       U10)
    DF(U11
       U12))
  N(DG(U13
       U14)
    DH(U15
       U16))))
```

- Partially factored:

```
(A(K(DA(U1)
     DA(U2)
     DB(U3
        U4))
   L(DC(U5
        U6))
   L(DD(U7
        U8)))
 B(M(DE(U9
        U10)
     DF(U11
        U12))
```

```
    N(DG(U13))
    N(DG(U14))
    N(DH(U15
         U16))))
```

- Unfactored, equivalent to the initial list:

```
(A(K(DA(U1)))
 A(K(DA(U2)))
 A(K(DB(U3)))
 A(K(DB(U4)))
 A(L(DC(U5)))
 A(L(DC(U6)))
 A(L(DD(U7)))
 A(L(DD(U8)))
 B(M(DE(U9)))
 B(M(DE(U10)))
 B(M(DF(U11)))
 B(M(DF(U12)))
 B(N(DG(U13)))
 B(N(DG(U14)))
 B(N(DH(U15)))
 B(N(DH(U16))))
```

In the above lists, *"("* represents *begin_dest_operands*, *begin_REN_list*, *begin_DGN_list*, or *begin_DEN_list*. *")"* represents *end_DEN_list*, *end_DGN_list*, *end_REN_list*, or *end_dest_operands*. (Inner parentheses have precedence over outer parentheses.)

Begin_Dest_Operands

Description: The *beginning_of_the_destination_operands* marks the beginning of the *destination_list*.

Format: Constant byte string; value is X′C35201′

Dest_RGN_List

Description: The *destination_RGN_list* associates one destination RGN with at least one destination REN.

Begin_REN_List

Description: The *beginning_of_the_destination_REN_list* marks the beginning of a list of one or more *dest_REN*(s).

Format: Constant byte string; value is X′C35301′

Dest_REN_List

Description: The *destination_REN_list* associates one destination REN with at least one destination DGN.

Begin_DGN_List

Description: The *beginning_of_the_destination_DGN_list* marks the beginning of a list of one or more *dest_DGN*(s).

Format: Constant byte string; value is X'C35401'

Dest_DGN_List

Description: The *destination_DGN_list* associates one *dest_DGN* with at least one *dest_DEN*.

Begin_DEN_List

Description: The *beginning_of_the_destination_DEN_list* marks the beginning of a list of one or more *dest_DEN*(s).

Format: Constant byte string; value is X'C35501'

End_DEN_List

Description: The *end_destination_DEN_list* marks the end of the list begun by the corresponding *begin_DEN_list*.

End_DGN_List

Description: The *end_destination_DGN_list* marks the end of the list begun by the corresponding *begin_DGN_list*.

End_REN_List

Description: The *end_destination_REN_list* marks the end of the list begun by the corresponding *begin_REN_list*.

End_Dest_Operands

Description: The *end_destination_operands* marks the end of the *destination_list*.

Dist_Server_Operands

Description: The *distribution_server_operands* structure contains the *server_prefix* and the *server_object*.

Presence Rule: This information occurs only in Dist_MU *type* TRANSPORT when *server_object_ind* =X'0001'.

Server_Prefix

Description: The *server_prefix* contains information associated with the *server_object*.

Server_Parms

Description: The *server_parameters* structure contains parameters passed by DS to the destination server. This structure is never sent, and is retired in FS2.

Format: Undefined byte string

DS_Suffix (FS1)

Description: The *distribution_services_suffix* contains no information and marks the end of the message unit.

Dist_Report_Operands

Description: The *distribution_report_operands* structure contains all the report information describing the condition of a particular distribution.

Presence Rule: This information occurs only when Dist_MU is of type REPORT.

Report_Operands

Description: The *report_operands* structure contains all information pertaining to the originator of the distribution and the detector of an exception.

Report_Correlation

Description: The *report_correlation* contains information that uniquely identifies a distribution being reported on.

Reported-On_Seqno

Description: The *reported-on_origin_sequence_number* is the sequence number of the distribution being reported on.

Format: Character string; each character represents the EBCDIC representation of one digit of the sequence number.

Byte	Content
0-1	LT header
2-5	Sequence number Note: Values range from X'F0F0F0F1' to X'F9F9F9F9'.

Reported-On_DTM

Description: The *reported-on_date-time* is the date and time the distribution was originated.

Byte	Content
0-1	LT header
	DATE
2-3	Year, in binary (e.g., 1989 is encoded as X'07C5')
4	Month of the year, in binary (values from 1 to 12 are valid)
5	Day of the month, in binary (values from 1 to 31 are valid)
	TIME
6	Hour of the day, in binary (values from 0 to 23 are valid)
7	Minute of the hour, in binary (values from 0 to 59 are valid)
8	Second of the minute, in binary (values from 0 to 59 are valid)
9	Hundredth of the second, in binary (values from 0 to 99 are valid)

Example:

The date-time encoding for 12:00 noon on 2 January 1988 is:

```
X'07C401020C000000'
  yyyyMMddHHmmsshh
```

Gen_SNADS_Report

Description: The *general_SNADS_report* contains the DS report applicable to each user specified in *specific_report* for which a *spec_SNADS_report* is not supplied.

Note: Older DSUs may generate both *gen_SNADS_report* and *gen_DIA_report* in a single MU. All DSUs are able to receive such MUs. However, DSUs may ignore *gen_DIA_report* if *gen_SNADS_report* is present. A sending DSU never generates both a DIA report and a DS report for multiple destinations.

Presence Rule: This information occurs when *gen_SNADS_type* = X'0001'.

Gen_SNADS_Type

Description: The *general_SNADS_type* indicates that a DS condition is being reported.

Format: Hexadecimal code

Byte	Content
0-4	LLIDF header
5-6	X'0001' DS report Note: Any other value indicates that this is not a *gen_SNADS_report*.

Gen_SNADS_Content

Description: The *general_SNADS_contents* contains information describing the condition being reported on.

Gen_SNADS_Cond_Code

Description: The *general_SNADS_condition_code* is the particular condition being reported on.

Format: Hexadecimal code

Byte	Content	
0-1	LT header	
2-3	X'0001'	routing exception
	X'0002'	unknown user name
	X'0003'	hop count exhausted
	X'0004'	format exception
	X'0005'	function not supported
	X'0006'	specific-server exception
	X'0007'	unknown resource name (specific server)
	X'0008'	invalid server parameters
	X'0009'	unknown resource name (destination agent)
	X'000C'	operator intervention (purging)
	X'000D'	user names lost
	X'000E'	resource not available
	X'000F'	system exception
	X'0010'	insufficient resource
	X'0011'	storage-medium exception
	X'0012'	REMU exception
	X'0013'	server object size incompatible with capacity level
	Note: All other values are reserved.	

Gen_DIA_Report

Description: The *general_DIA_report* describes an application-layer condition. The *gen_DIA_report* applies to all users specified in *specific_report*. The interaction between *gen_DIA_report* and *spec_DIA_report* is defined by DIA.

Note: Older DSUs may generate both *gen_SNADS_report* and *gen_DIA_report* in a single MU. All DSUs can receive such MUs. However, DSUs may ignore *gen_DIA_report* if *gen_SNADS_report* is present. A sending DSU never generates both a DIA report and a DS report for multiple destinations.

Presence Rule: This information occurs when *gen_DIA_type* ≠ X'0001'.

Gen_DIA_Type

Description: The *general_DIA_type* indicates the type of DIA condition being reported.

Format: Hexadecimal code

Byte	Content
0-4	LLIDF header
5-6	X′0001′ indicates this is not a *gen_DIA_report* X′0200′ DIA application exceptions X′FEFF′ reserved for 5520 migration Note: All other values are reserved.

Gen_DIA_Contents

Description: The *general_DIA_contents* structure contains a DIA-defined byte string.

Length Restriction: Older DSUs may generate MUs with length of up to 517. All DSUs receive such MUs without generating an exception. However, DSUs may modify such MUs to force the length to be 69 or less. For *gen_DIA_type* of X′0200′ (DIA application exceptions), the truncation algorithm is given in the ***DIA Transaction Programmer's Guide***. The length is at least 7, since *gen_DIA_contents* contains at least a null LT (an LT of length 2).

Format: Undefined byte string

Specific_Report

Description: The *specific_report* contains the portion of the destination users that are being reported on. Any specific DS and/or DIA reports are also specified within this structure.

Begin_Report_DGN_List

Description: The *beginning_of_report_DGN_list* marks the beginning of the *specific_report*.

Format: Constant byte string; value is X′C35401′

Report_DGN_List

Description: The *report_DGN_list* associates one *reported-on_dest_DGN* with at least one *reported-on_dest_DEN*.

Begin_Report_DEN_List

Description: The *beginning_of_report_DEN_list* marks the beginning of a list of one or more *reported-on_dest_DEN*s.

Format: Constant byte string; value is X′C35501′

Report_DEN_List

Description: The *report_DEN_list* associates one *reported-on_dest_DEN* with a specific DS and/or DIA report.

Spec_SNADS_Report

Description: The *specific_SNADS_report* is a report on one particular user. This report overrides the *gen_SNADS_report*, if one exists, for that particular user.

Note: Older DSUs may generate both *spec_SNADS_report* and *spec_DIA_report* in a single MU. All DSUs can receive such MUs. However, DSUs may ignore *spec_DIA_report* if *spec_SNADS_report* is present. A sending DSU never generates both a DIA report and a DS report for multiple destinations.

Presence Rule: This information occurs when *spec_SNADS_type* = X′0001′.

Spec_SNADS_Type

Description: The *specific_SNADS_type* indicates that a DS condition is being reported.

Format: Hexadecimal code

Byte	**Content**
0-4	LLIDF header
5-6	X′0001′ DS report Note: Any other value indicates that this is not a *spec_SNADS_report*.

Spec_SNADS_Cont

Description: The *specific_SNADS_contents* contains information describing a condition being reported on.

Spec_SNADS_CC

Description: The *specific_SNADS_condition_code* describes the particular condition being reported on.

Format: Hexadecimal code

Byte	Content	
0-1	LT header	
2-3	X'0001'	routing exception
	X'0002'	unknown user name
	X'0003'	hop count exhausted
	X'0004'	format exception
	X'0005'	function not supported
	X'0006'	specific-server exception
	X'0007'	unknown resource name (specific server)
	X'0008'	invalid server parameters
	X'0009'	unknown resource name (destination agent)
	X'000C'	operator intervention (purging)
	X'000D'	user names lost
	X'000E'	resource not available
	X'000F'	system exception
	X'0010'	insufficient resource
	X'0011'	storage-medium exception
	X'0012'	REMU exception
	X'0013'	server object size incompatible with capacity level
	Note: All other values are reserved.	

Spec_DIA_Report

Description: The *specific_DIA_report* describes a DIA-specific report on one particular user.

Note: Older DSUs may generate both *spec_SNADS_report* and *spec_DIA_report* in a single MU. All DSUs can receive such MUs. However, DSUs may ignore *spec_DIA_report* if *spec_SNADS_report* is present. A sending DSU never generates both a DIA report and a DS report for multiple destinations.

Presence Rule: This information occurs when *spec_DIA_type* ≠ X'0001'.

Spec_DIA_Type

Description: The *specific_DIA_type* indicates the type of DIA condition being reported.

Format: Hexadecimal code

Byte	Content
0-4	LLIDF header
5-6	X'0001' indicates this is not a *spec_DIA_report* X'0200' DIA application exceptions X'FEFF' reserved for 5520 migration Note: All other values are reserved.

Spec_DIA_Contents

Description: The *specific_DIA_contents* structure contains a DIA-defined byte string.

Length Restriction: Older DSUs may generate MUs with length of up to 517. All DSUs receive such MUs without generating an exception. However, DSUs may modify such MUs to force the length to be 69 or less. For *spec_DIA_type* of X'0200' (DIA application exceptions), the truncation algorithm is given in the ***DIA Transaction Programmer's Guide***. The length is at least 7, since *spec_DIA_contents* contains at least a null LT (an LT of length 2).

Format: Undefined byte string

End_Report_DEN_List

Description: The *end_report_DEN_list* marks the end of the list begun by *begin_report_DEN_list*.

End_Report_DGN_List

Description: The *end_report_DGN_list* marks the end of the *specific_report*.

Sender_Exception_MU (Type FS1)

Description: The *sender_exception_MU* (type FS1) is sent from the sender to the receiver when the sender detects an exception while sending a Dist_MU.

Format: Byte string

Byte	Bit	Content	
0-4		LLIDF header	
5		Severity:	
	0-1	11	catastrophic
		Class:	
	2-7	000101	sender
6		Exception condition code:	
		X′06′	execution terminated
		X′0B′	I/O error
		X′0F′	length invalid
		X′18′	content error
7		Exception object:	
		X′01′	IU prefix
		X′07′	command
		X′0C′	document unit
		X′13′	IU suffix
		X′17′	unknown subfield
		X′1A′	distribution object prefix
		X′1B′	distribution object data

Note: Other values and their corresponding meanings are represented under *receiver_exception_code.*

Receiver_Exception_MU (Type FS1)

Description: The *receiver_exception_MU* (type FS1) is sent from the receiver to the sender when the receiver detects an exception while receiving a Dist_MU.

Receiver_Exception_Command

Description: The *receiver_exception_command* contains all information used for identifying the exception that occurred.

Receiver_Exception_Correl

Description: The *receiver_exception_correlation* contains the *prefix* ID value from the rejected Dist_MU.

Format: Byte string

Byte	Content
0-4	LLIDF header
5	Correlation field: X′00′ Note: All other values are reserved.
6	Command sequence number: X′01′ Note: All other values are reserved.
7-22	Correlation MU ID; value from the *prefix* of the Dist_MU

Exception_And_Reply_Data

Description: The *exception_and_reply_data* contains information pertaining to the exception causing the rejection of the Dist_MU.

Receiver_Exception_Code

Description: The *receiver_exception_code* identifies the type of exception encountered and, conditionally, the portion of the Dist_MU containing the exception.

Format: Byte string

Byte	Bit	Content	
0-4		LLIDF header	
5		Severity:	
	0-1	11	catastrophic
		Note: All other values for bits 0-1 are reserved.	
		Class:	
	2-7	000010	syntactic
		000011	semantic
		000100	process
		Note: All other values for bits 2-7 are reserved or defined elsewhere.	
6		Exception condition code (indicates reason for exception):	
		X'01'	function not supported
		X'02'	data not supported
		X'04'	resource not available
		X'06'	execution terminated
		X'07'	data not found
		X'08'	segmentation
		X'0A'	sequence
		X'0B'	I/O error
		X'0C'	ID invalid
		X'0E'	format invalid
		X'0F'	length invalid
		X'10'	indicator invalid
		X'11'	range exceeded
		X'15'	subfield length invalid
		X'16'	subfield type invalid
		X'17'	invalid parameters
		X'18'	content error
		Note: All other values are reserved.	
		Exception object	
7		(indicates the syntactical entity in error):	
		X'01'	IU prefix
		X'02'	IU identifier
		X'07'	command
		X'08'	command operand
		X'09'	operand value
		X'0C'	document unit
		X'0D'	document unit identifier

Byte	Bit	Content
		X'0E' document profile
		X'0F' document profile parameter
		X'10' document content introducer
		X'11' document content control
		X'12' document content data
		X'13' IU suffix
		X'14' segment
		X'16' unsupported subfield
		X'17' unknown subfield
		X'1A' distribution object prefix
		X'1B' distribution object data
		Note: All other values are reserved.
8-254		Exception data contains the Dist_MU structures in error

Reply_Data

Description: The *reply_data* describes which DSU rejected the Dist_MU and why the Dist_MU was rejected.

SNADS_Report

Description: The *SNADS_report* contains information describing the particular DS exception that caused the Dist_MU to be rejected.

SNADS_Report_Type

Description: The *SNADS_report_type* indicates that a DS exception is being reported.

Format: Hexadecimal code

Byte	Content
0-4	LLIDF header
5-6	X'0001' DS report Note: Any other value indicates that this is not a *SNADS_report*.

SNADS_Report_Cont

Description: The *SNADS_report_contents* structure contains information describing the type of DS condition in the Dist_MU.

SNADS_Report_CC

Description:	The *SNADS_report_condition_code* describes the particular DS condition that caused the Dist_MU to be rejected.
Format:	Hexadecimal code

Byte	Content	
0-1	LT header	
2-3	X'0001'	routing exception
	X'0002'	unknown user name
	X'0003'	hop count exhausted
	X'0004'	format exception
	X'0005'	function not supported
	X'0006'	specific-server exception
	X'0007'	unknown resource name (specific server)
	X'0008'	invalid server parameters
	X'0009'	unknown resource name (destination agent)
	X'000E'	resource not available
	X'000F'	system exception
	X'0010'	insufficient resource
	X'0011'	storage-medium exception
	X'0013'	server object size incompatible with capacity level
	Note: All other values are reserved.	

DIA_Report

Description:	The *DIA_report* describes a DIA condition being reported.
Note:	When generating a Dist_MU *type* REPORT with report information supplied by a REMU (type FS1), the reporting DSU may ignore *DIA_report.*
Presence Rule:	This information occurs when *gen_DIA_type* ≠ X'0001'.

DIA_Report_Type

Description:	The *DIA_report_type* indicates the type of DIA condition being reported.
Format:	Hexadecimal code

Byte	Content	
0-4	LLIDF header	
5-6	X'0001'	indicates this is not a *DIA_report*
	X'0200'	DIA application exceptions
	X'FEFF'	reserved for 5520 migration
	Note: All other values are reserved.	

DIA_Report_Cont

Description:	The *DIA_report_contents* structure contains a DIA-defined byte string.
Format:	Undefined byte string

Graphic Character Sets 1134 and 930

Table 84 (Page 1 of 2). Graphic Character Sets 1134 and 930

Hex Code	Gra-phic	Description	Set	
			1134	930
40		Space		X
4B	.	Period		X
50	&	Ampersand		X
59		Sharp s		X
5B	$	Dollar Sign		X
60	-	Minus Sign		X
61	/	Slash		X
62		A Circumflex, Capital		X
63		A Diaeresis, Capital		X
64		A Grave, Capital		X
65		A Acute, Capital		X
66		A Tilde, Capital		X
67		A Overcircle, Capital		X
68		C Cedilla, Capital		X
69		N Tilde, Capital		X
6B	,	Comma		X
71		E Acute, Capital		X
72		E Circumflex, Capital		X
73		E Diaeresis, Capital		X
74		E Grave, Capital		X
75		I Acute, Capital		X
76		I Circumflex, Capital		X
77		I Diaeresis, Capital		X
78		I Grave, Capital		X
7B	#	Number Sign		X
7C	@	At Sign		X
7D	’	Apostrophe		X
80		O Slash, Capital		X
81	a	a, Small		
82	b	b, Small		
83	c	c, Small		
84	d	d, Small		
85	e	e, Small		
86	f	f, Small		
87	g	g, Small		
88	h	h, Small		
89	i	i, Small		
91	j	j, Small		

Table 84 (Page 1 of 2). Graphic Character Sets 1134 and 930

Hex Code	Gra-phic	Description	Set	
			1134	930
92	k	k, Small		
93	l	l, Small		
94	m	m, Small		
95	n	n, Small		
96	o	o, Small		
97	p	p, Small		
98	q	q, Small		
99	r	r, Small		
9A		a Underscore, Small		X
9B		o Underscore, Small		X
9E		AE Dipthong, Capital		X
A0		Micro, Mu		X
A2	s	s, Small		
A3	t	t, Small		
A4	u	u, Small		
A5	v	v, Small		
A6	w	w, Small		
A7	x	x, Small		
A8	y	y, Small		
A9	z	z, Small		
AC		D Stroke, Capital		X
AD		Y Acute, Capital		X
AE		Thorn, Capital		X
C1	A	A, Capital	X	X
C2	B	B, Capital	X	X
C3	C	C, Capital	X	X
C4	D	D, Capital	X	X
C5	E	E, Capital	X	X
C6	F	F, Capital	X	X
C7	G	G, Capital	X	X
C8	H	H, Capital	X	X
C9	I	I, Capital	X	X
D1	J	J, Capital	X	X
D2	K	K, Capital	X	X
D3	L	L, Capital	X	X
D4	M	M, Capital	X	X
D5	N	N, Capital	X	X
D6	O	O, Capital	X	X

Table 84 (Page 2 of 2). Graphic Character Sets 1134 and 930

Hex Code	Graphic	Description	Set	
			1134	930
D7	P	P, Capital	X	X
D8	Q	Q, Capital	X	X
D9	R	R, Capital	X	X
DF		y Diaeresis, Small		X
E2	S	S, Capital	X	X
E3	T	T, Capital	X	X
E4	U	U, Capital	X	X
E5	V	V, Capital	X	X
E6	W	W, Capital	X	X
E7	X	X, Capital	X	X
E8	Y	Y, Capital	X	X
E9	Z	Z, Capital	X	X
EB		O Circumflex, Capital		X
EC		O Diaeresis, Capital		X
ED		O Grave, Capital		X
EE		O Acute, Capital		X
EF		O Tilde, Capital		X
F0	0	Zero	X	X
F1	1	One	X	X
F2	2	Two	X	X
F3	3	Three	X	X
F4	4	Four	X	X
F5	5	Five	X	X
F6	6	Six	X	X
F7	7	Seven	X	X
F8	8	Eight	X	X
F9	9	Nine	X	X
FB		U Circumflex, Capital		X
FC		U Diaeresis, Capital		X
FD		U Grave, Capital		X
FE		U Acute, Capital		X

Note: Character set A, CGCSGID (00961-00500), is a superset of character set 1134. Character set A also contains ″$″ (X′5B′), ″#″ (X′7B′) and ″@″ (X′7C′). New DS implementations are able to receive structures using character set A.

Character set AE, CGCSGID (01130-00500), is a superset of character set 1134. Character set AE also contains ″.″ (X′4B′), ″$″ (X′5B′), ″#″ (X′7B′), ″@″ (X′7C′), and all lower-case alphabetics (a-z).
New DS implementations are able to receive structures using character set AE.

Transaction Program and Server Names

Following is a list of all transaction program and server names defined for SNA/DS, in the FM header 5 (Attach), in the Distribution MU, or used internally in the distribution service unit (DSU).

Code	Meaning
X'20F0F0F0'	DIA process destination transaction program name
X'20F0F0F1'	DIA server name
X'20F0F0F2'	DIASTATUS transaction program name
X'21F0F0F1'	DS_SEND transaction program name (FS1)
X'21F0F0F2'	DS_RECEIVE transaction program name (FS1)
X'21F0F0F3'	DS_ROUTER_DIRECTOR transaction program name
X'21F0F0F6'	SNA/DS general server name
X'21F0F0F7'	DS_SEND transaction program name (FS2)
X'21F0F0F8'	DS_RECEIVE transaction program name (FS2)
X'23F0F0F0'	SNA/MS Change Management agent TP name
X'24F0F0F0'	SNA/File Services server name
X'30F0F0F2'	Object Distribution transaction program for IBM System 36 and System 38.
X'30F0F0F3'	Object Distribution server transaction program for IBM System 36 and System 38.

Code Points Used by SNA/DS FS2

The values of the ID component of the LLID structure as used for SNA/DS GDS variables are shown below:

ID	Structure Name
1532	SNA Condition Report
1570	Transport Prefix
1571	Transport Command
1572	Destination List
1573	Agent Object
1574	Server Object
1575	Report Command
1576	Report Information
1577	Receiver Exception Command
1578	Sender Exception Message Unit (type FS2)
1579	Completion Query Message Unit
157A	Completion Report Message Unit
157B	Continuation Prefix
157C	Report Prefix
157E	Purge Report Message Unit
157F	Suffix
1580	Supplemental Distribution Info2
1582	Reported-On Supplemental Distribution Info2
1583	Report-To DSU/User
1585	Reset Request Message Unit
1586	Reset Accepted Message Unit

Code Points Used by SNA/DS FS1

The values of the ID component of the LLIDF structure as used for SNA/DS GDS variables are shown below:[2]

ID	Structure Name
C001*	In DIA, MU PREFIX; in DS, Prefix within DIST_MU or within REMU (type FS1)
C101*	in DIA, MU CMD NO REPLY ACKNOWLEDGE; in DS, Command within REMU (type FS1)
C105	Command, DIST_MU
C322*	in DIA, MU OPERAND IMM DATA EXCEPTION-CODE; in DS, Exception Code, within REMU (type FS1)
C328*	in DIA, MU OPERAND IMM DATA DATA CORRELATION; in DS, Correlation, within REMU (type FS1)
C32D*	in DIA, MU OPERAND IMM DATA USER-DATA; in DS, Agent Object within DIST_MU
C33D*	in DIA, MU OPERAND IMM DATA STATUS-INFORMATION; in DS, Distribution General Options, within DIST_MU
C340*	in DIA, MU OPERAND IMM DATA DISTRIBUTION-IDENTIFIER; in DS, Distribution Identifier, within DIST_MU
C343*	in DIA, MU OPERAND IMM DATA GENERAL-ROUTING-DATA; in DS, Report-To Options within DIST_MU
C345*	in DIA, MU OPERAND IMM DATA REPLY DATA; in DS, Reply Data, within REMU (type FS1)
C350	Beginning of Destination Operand Lists, of the Specific Report Lists, within DIST_MU
C351	End of Destination Operands Lists, of the Specific Report Lists, within DIST_MU
C352	Routing Group Name (RGN) of Destination Operands, within DIST_MU
C353	Routing Element Name (REN) of REN List, within DIST_MU
C354	Distribution Group Name (DGN) of DGN List, within DIST_MU
C355	Distribution Element Name (DEN) of DEN List, within DIST_MU
C356	Report Type, within DIST_MU
C357	Report Contents, within DIST_MU
C360	Report-To Address, within DIST_MU
C361	Receiving DSU, within DIST_MU or within REMU (type FS1)

[2] The asterisk following the ID indicates that that identifier is used by both DIA (Document Interchange Architecture) and DS.

C908 Server Object, within DIST_MU

C90A Server Prefix, within DIST_MU

CF01* in DIA, MU SUFFIX NORMAL-TERMINATION; in DS, Suffix within DIST_MU or within REMU (type FS1)

CF02* in DIA, MU SUFFIX ABNORMAL-TERMINATION; in DS, SEMU (type FS1)

Terminology Mappings

Table 85 (Page 1 of 3). Terminology Mappings

FS2 TERMINOLOGY	Current FS1 TERMINOLOGY	Old FS1 TERMINOLOGY
Dist_Transport_MU	Dist_MU (type Transport)	Dist_IU (type Data)
Transport_Prefix	Prefix	Prefix
Hop_Count	Hop_Count	Dist_Dest_Hops
MU_ID	—	—
Transport_Command	Dist_Command	Dist_CMD
Dist_Flags	Dist_Flags (FS1)	Dist_Flags
Service_Parms	Service_Parms	DSL
Server_Obj_Byte_Count	Server_Obj_Byte_Count	Data_Size
Origin_Agent	Origin_Agent	Dest_TPN
Server	Server	Server_Name
Origin_DSU	—	—
Origin_RGN	Origin_RGN	Orig_RGN
Origin_REN	Origin_REN	Orig_REN
Origin_User	—	—
Origin_DGN	Origin_DGN	Orig_DGN
Origin_DEN	Origin_DEN	Orig_DEN
Seqno_DTM	Origin_Seqno, Origin_DTM	Orig_Seqno, Orig_DTM
Supplemental_Dist_Info1	—	—
Agent_Correl	Agent_Correl	Orig_Correl
Report-To_DSU	—	—
Report-To_RGN	Report-To_RGN	Fdbk_RGN
Report-To_REN	Report-To_REN	Fdbk_REN
Report-To_User	—	—
Report-To_DGN	Report-To_DGN	Fdbk_DGN
Report-To_DEN	Report-To_DEN	Fdbk_DEN
Report_Service_Parms	Report_Service_Parms	Fdbk_DSL
Report-To_Agent	Report-To_Agent	Fdbk_TPN
Dest_Agent	—	—
Unrecognized_Reserve	—	—
Dest_List	Destination_Operands	Destination_Operands
Dest	—	—
Dest_DSU	—	—
Dest_RGN	Dest_RGN	Dest_RGN
Dest_REN	Dest_REN	Dest_REN
Dest_User	—	—
Dest_DGN	Dest_DGN	Dest_DGN
Dest_DEN	Dest_DEN	Dest_DEN
Agent_Object	Agent_Object	Dest_Appl_Parms
Server_Object	Server_Object	Distrib_Object_Data
Supplemental_Dist_Info2	—	—

Table 85 (Page 2 of 3). Terminology Mappings

FS2 TERMINOLOGY	Current FS1 TERMINOLOGY	Old FS1 TERMINOLOGY
DS_Suffix	DS_Suffix	Suffix
Dist_Report_MU	Dist_MU (type Report)	Dist_IU (type Status)
Report_Prefix	—	—
Report_Command	—	—
Reporting_DSU	—	—
Reporting_RGN	—	—
Reporting_REN	—	—
Report_DTM	—	—
Report-To_DSU_User	—	—
Report_Information	—	—
Reported-On_Origin_DSU	—	—
Reported-On_Origin_RGN	—	—
Reported-On_Origin_REN	—	—
Reported-On_Origin_User	—	—
Reported-On_Origin_DGN	Reported-On_Origin_DGN	Orig_DGN
Reported-On_Origin_DEN	Reported-On_Origin_DEN	Orig_DEN
Reported-On_Seqno_DTM	Reported-On_Seqno, Reported-On_DTM	Orig_Seqno, Orig_DTM
Reported-On_Supp_Dist_Info1	—	—
Reported-On_Supp_Dist_Info2	—	—
Reported-On_Agent_Correl	Reported-On_Agent_Correl	Orig_Correl
Reported-On_Dest_Agent	—	—
SNA_Condition_Report	—	—
SNA_Report_Code	—	—
Structure_Report	—	—
Structure_State	—	—
Structure_Contents	—	—
Parent_Spec	—	—
Parent_ID_Or_T	—	—
Parent_Class	—	—
Parent_Position	—	—
Parent_Instance	—	—
Structure_Spec	—	—
Structure_ID_Or_T	—	—
Structure_Class	—	—
Structure_Position	—	—
Structure_Instance	—	—
Structure_Segment_Num	—	—
Structure_Byte_Offset	—	—
Sibling_List	—	—
Reported-On_Dest_List	Specific_Report	Specific_Status
Reported-On_Dest_Pfx	—	—
Reported-On_Dest	—	—

Table 85 (Page 3 of 3). Terminology Mappings

FS2 TERMINOLOGY	Current FS1 TERMINOLOGY	Old FS1 TERMINOLOGY
Reported-On_Dest_DSU	—	—
Reported-On_Dest_RGN	—	—
Reported-On_Dest_REN	—	—
Reported-On_Dest_User	—	—
Reported-On_Dest_DGN	Reported-On_Dest_DGN	Stat_DGN
Reported-On_Dest_DEN	Reported-On_Dest_DEN	Stat_DEN
Reported-On_Dest_Sfx	—	—
Supplemental_Report	—	—
Dist_Continuation_MU	—	—
Continuation_Prefix	—	—
Restarting_Byte_Position	—	—
Sender_Exception_MU	Sender_Exception_MU	Suffix (type 2)
Receiver_Exception_MU	Receiver_Exception_MU	Ack_IU
Receiver_Exception_Command	Receiver_Exception_Command	Ack_Cmd
Sender_Retry_Action	—	—
Receiving_DSU	Receiving_DSU	Rcv_DSUN
Receiving_RGN	Receiving_RGN	Rcv_DSUN_RGN
Receiving_REN	Receiving_REN	Rcv_DSUN_REN
Completion_Query_MU	—	—
Completion_Report_MU	—	—
Indicator_Flags	—	—
Last_Structure_Received	—	—
Last_Byte_Received	—	—
Purge_Report_MU	—	—
Reset_Request_MU	—	—
Reset_DTM	—	—
Reset_Accepted_MU	—	—

Glossary

Glossary terms are defined as they are used in this book. If you cannot find a term in this glossary, refer to the index or to the glossary in *SNA Technical Overview,* GC30-3073.

A

Agent. An application program that submits and receives information to and from the distribution service, across the agent protocol boundary.

Agent Object. Small amounts of data submitted for distribution by the origin agent. The agent object is stored and distributed by DS and passed directly to the destination agent. Larger amounts of data, or data that requires a particular kind of handling (encryption, for example, or specialized parsing) usually flow in the server object.

Agent Protocol Boundary. The logical interface between agents or operators and SNA/DS.

Alternate Routing. An implementation-defined mechanism that provides an alternate route in the routing table when connections are unavailable for a particular combination of destination DSU and service parameters.

Auxiliary Server Operation. The copy-making steps performed when transferring the server object between the specific server and the general server.

B

Bilingual DSU. A DSU that acts as a gateway between FS1-only portions of the DS network and FS2-supporting portions, translating between the format sets as necessary.

Byte-Count Restart. A type of Mid-MU restart whereby the transmission of the server object can be restarted at any byte position.

C

Capacity Service Parameter. The service parameter that describes the required storage capacity for the distribution.

Completion Query Message Unit (CQMU). A control message unit sent by the sending DSU to query the completion status of a particular message unit at the receiving DSU.

Completion Report Message Unit (CRMU). A control message unit sent by the receiving DSU to report on the completion status of a particular message unit.

Connection. In SNA/DS, the set of concurrent conversations with a particular partner LU using a particular mode name. A connection may consist of one or more than one conversation.

Connection-Oriented Service. A communications service that provides a direct connection between the communicating parties.

Connectionless Service. A service that performs communication without the establishment of a direct connection between (or among) the communicating parties.

Control Message Unit (CMU). A type of message unit containing information about a conversation or about a DMU. The control information is used strictly by adjacent DSUs to manage traffic between the DSUs. CQMUs, CRMUs, PRMUs, RAMUs, REMUs, RRMUs, and SEMUs are referred to collectively as control message units.

D

Default Directing. The use of default "tokens" (the * in this documentation) in place of parts of the user name. Default directing alleviates the need to specify every user name or every agent name in every local directory throughout the network.

Default Routing. The use of default "tokens" (the * in this documentation) in place of parts of the DSU name. Default routing alleviates the need to specify every DSU name in every routing table throughout the network.

Destination. One of the intended recipients of a distribution. A destination may be either a user or DSU (node).

Destination DSU. A DSU at which one or more recipients of a distribution reside.

Destination Server. A server used at the destination DSU to store the server object in an application's private storage space.

Direct Fetch. An implementation elective that bypasses the auxiliary server operation from the specific server to the general server. The outgoing server object is directly retrieved using the specific server at send time.

Direct Store. An implementation elective that bypasses the auxiliary server operation from the general server to the specific server. The incoming server object is directly stored into an application's (or user's) private space at receive time.

Directing Sublayer. The DS sublayer responsible for associating a DSU name with every destination user name in a distribution and for determining, at the destination(s) of a distribution, local delivery information for each destination.

Distribution. (1) In general, the function of transporting information from an origin to one or more destinations in a DS network; a distribution is the result of a specific request for distribution service. (2) A synonym for the actual data transported by DSUs in honoring such a request; see distribution message unit (DMU).

From the perspective of the user or agent, a distribution is the complete set of results of a request. It may sometimes be appropriate from that perspective to refer to the parts of the distribution flowing along different routes as distribution copies.

From the perspective of a particular Distribution Service Unit (DSU), a distribution may be complete or it may be only a copy of a larger distribution. From that DSU's perspective every distribution copy being processed through it is referred to as a distribution. A receiving DSU cannot determine whether or not it has received the only copy of a particular distribution. Therefore, every distribution copy being processed by that DSU is referred to as a distribution.

Distribution Continuation Message Unit (DCMU). A distribution message unit used by a sending DSU to continue transmission of a suspended message unit.

Distribution Copy. The result of transmitting a distribution. Each distribution copy contains some or all of the complete set of destinations for the entire distribution.

Distribution Element Name (DEN). The second part of a distribution-user name. The user element name (DEN) is unique within its particular group (DGN).

Distribution Group Name (DGN). The first part of a distribution-user name. The distribution group name (DGN) is unique throughout the DS network. The DGN is intended to be a convenient and natural grouping of names, with no reference to a location.

Distribution Identification. The collection of structures in the DTMU that uniquely identifies a distribution in the DS network. The *dist_ID* consists of the origin DSU name, origin user name (if any), origin agent name, a sequence number, and a date.

Distribution Message Unit (DMU). A type of message unit used to convey distribution information between DSUs. DTMUs, DRMUs, and DCMUs are referred to collectively as distribution message units.

Distribution Report. Information, provided in a DRMU by the detecting DSU in the DS network, on the condition of a distribution.

Distribution Report Message Unit (DRMU). A distribution message unit containing a distribution report.

Distribution Services (DS). See SNA/Distribution Services.

Distribution Service Unit (DSU). The collection of distribution transaction programs and data structures (e.g., queues, routing tables, user directory) that provide the distribution service at a particular location in a DS network.

Distribution Transport Message Unit (DTMU). A distribution message unit used to deliver the information given in a distribution request to the specified destinations.

Distribution Transport Sublayer. The DS sublayer responsible for sending and receiving MUs between adjacent DSUs. This sublayer manages the LU 6.2 conversations and provides encoding and decoding of MUs.

DSU Name. The name of a distribution service unit. The DSU name consists of two parts, the routing group name and the routing element name. It identifies a specific location in the DS network. Users typically are not aware of DSU names (i.e., RGNs and RENs).

DSU Role. The function performed by a DSU while processing a particular distribution. A DSU may play any of several roles, depending upon the distribution (e.g., the DSU at which a distribution is originated plays the role of the origin; a DSU that receives a distribution and forwards it to another DSU plays the role of an intermediate DSU).

E

Early Acceptance. The process by which a specific server reports a partially successful server operation to the local agent, and allows DS to process the remainder of the distribution normally.

Elective. An implementation choice as to how or when a function is provided, made for performance or development cost reasons. All permissible electives are defined by the architecture, since the effects of an elective are observable outside the DSU.

Exception Hold. The process of holding the next-DSU queue to stop transmission to the adjacent DSU, because an exception condition was encountered. The hold can be released by the operator or by the initiation of a new instance of DS_Send.

F

Fan-Out. The process of creating copies of a distribution to be delivered locally or to be sent through the network.

Format Set 1 (FS1). The earlier version of the DS formats and protocols.

Format Set 2 (FS2). The current version of the DS formats and protocols.

G

General Server. A server that DS uses to store and retrieve server objects in DS storage space.

I

Intermediate DSU. A DSU through which a distribution passes on its way from the origin DSU to the destination DSU(s). An intermediate DSU receives, routes, and may redirect the distribution before forwarding it.

L

LLID. The introducer of a length-bounded encoding structure whose identifier is two bytes long.

LLID Restart. A type of Mid-MU restart whereby the transmission is resumed at the beginning of an LLID structure in the DMU.

Local Reports. Information provided to the agent on a condition that occurred before DS accepted responsibility for the distribution request or when DS was performing some application-specific operation (e.g., a specific server operation). Local reports are delivered to the agent across the agent protocol boundary.

Location-independent. The property of a user name whereby it does not depend on the DSU at which the user resides. This provides the ability to move users from one DSU to another without changing their user names.

LT. The introducer of a length-bounded encoding structure whose identifier is one byte long.

LU 6.2 Conversation. In SNA, a logical connection between two transaction programs using an LU 6.2 session. Conversations are delimited by brackets to gain exclusive use of a session.

LU 6.2 Protocol Boundary. The logical interface between LU 6.2 and DS across which information is passed.

M

Message Unit (MU). In general, an architecturally defined structure for information communicated from one process to another. In DS, the entity that flows between distribution service units.

Mid-MU Restart. The capability of restarting a failed transmission at or near the point of failure, rather than retransmitting from the beginning.

N

Node Destination. See Destination.

O

Operator. A person or program that interacts with the DSU by issuing commands to perform such tasks as system or network maintenance functions.

Operator Hold. The process of holding a distribution or the next-DSU queue in response to operator action.

Option Set. An architecturally defined group of DS functions. All DS implementations support the base set of DS functions, and may choose which option sets to support, if any.

Origin DSU. The DSU at which the originator of a distribution resides.

Origin server. The server, named in the distribution request, from which DS is to obtain the server object to distribute.

Other-Assignment. The name given to the action of assigning read access to the server object, whenever the issuer of the Assign_Read_Access verb specifies a different value for the *assigning_process* and *assigned_process*.

P

Presentation Services Sublayer. The DS sublayer with which agents and operators interact.

Priority Service Parameter. The service parameter used to specify the relative urgency of a particular distribution.

Protection Service Parameter. The service parameter used to specify whether the distribution must be stored on non-volatile storage while a DSU has responsibility for it.

Protocol Boundary. A logical interface that defines the interactions between DS and another entity. The boundary defines the functions provided by and expected by the entities on either side of the boundary.

Protocol Boundary Verb. A logical command or request issued across a protocol boundary.

Purge Report Message Unit (PRMU). A control message unit transmitted by the sending DSU to inform the receiving DSU that the sender has marked a particular MU_ID as PURGED in its MU_ID registry.

Q

Queue Protocol Boundary. The logical interface between the queue manager and DS across which information is passed.

R

Receive Time. The period of time required by a DSU to receive a DMU.

Receiving Sequence. A sequence of verbs issued by an agent to receive a distribution.

Receiver Exception Message Unit (REMU). A control message unit transmitted by the receiving DSU to the sending DSU when the receiver detects an exception while receiving a DMU.

Redirection. The process whereby DS receives a distribution, changes one or more DSU names (for one or more destinations), and forwards the distribution.

Report Service Parameters. The parameters supplied by the origin agent to specify the level of service requested for the distribution report. The origin agent supplies report service parameters when it wants to override the service parameters that would be routinely generated for the report by the reporting DSU.

Report-to Agent. The agent, specified by the originator, that will be invoked to receive the distribution report. If an agent is not explicitly defined, the origin agent will be invoked to receive the distribution report.

Reset Accepted Message Unit (RAMU). A control message unit sent by the receiving DSU, in response to an RRMU, to inform the sending DSU that the MU_ID registry at the receiving DSU has been reset.

Reset Request Message Unit (RRMU). A control message unit sent by the sending DSU to request the receiving DSU to reset its MU_ID registry.

Responsible DSU. For a distribution on which reporting has been requested, the DSU that must generate the distribution report in the event of an exception.

Reversible Server. A server that can store an object and later retrieve a byte-perfect copy of the object.

Routing Element Name (REN). The second part of the two-part DSU name. The routing element name (REN) must be unique within its particular group (RGN).

Routing Group Name (RGN). The first part of the two-part DSU name. This is typically, but not necessarily, the network ID.

Routing Sublayer. The DS sublayer that determines where to send distributions based on the DSU names in the destination list. For local destinations, the distribution is passed to the directing sublayer; for remote destinations, the distribution is placed on the appropriate next-DSU queue(s).

Routing Table. A table which defines the connection to be used to forward a distribution to a particular destination at a particular level of service.

S

Security Service Parameter. The service parameter used to specify whether the distribution is to be routed through the SNA/DS network using only sessions defined as secure.

Self-Assignment. The name given to the action of assigning read access to the server object, whenever the issuer of the Assign_Read_Access verb specifies the same value for the *assigning_process* and *assigned_process*.

Send Time. The period of time required by a DSU to send a DMU.

Sender Exception Message Unit (SEMU). A control message unit transmitted by the sending DSU to inform the receiving DSU that the sending DSU has encountered an exception while sending a DMU.

Sending Sequence. A sequence of verbs issued by an agent to originate a distribution.

Server. The process that fetches and stores server objects and controls access to them.

Server Access. Information provided to a server that is used to access the server object.

Server Object. Data specified by the originator of the distribution that is to be retrieved and stored using a server.

Server Protocol Boundary. The logical interface between servers and the distribution service across which distribution objects are passed.

Service Parameters. The parameters used to map particular types of DS traffic to particular classes of service offered by the lower layers of SNA.

SNA Condition Report (SNACR). A standard structure used by SNA components to report exception information. The structure contains the report code describing the condition and may provide additional supporting information.

SNA/Distribution Services (SNA/DS). A connectionless communications service that distributes objects over a network of LU 6.2 connections.

Specific Server. A server that stores objects into and fetches objects from an application's private storage space. A specific server may be sensitive to the contents of the byte streams it processes.

Specific Server Information. Instructions provided for the specific server to build and process the object prior to presenting it to DS.

T

Transaction Programs. In DS, the processes that provide the distribution service, as well as the agents and servers with which the distribution service interacts.

U

User Destination. See Destination.

User Directory. From the perspective of a particular distribution service unit (DSU), a local table of entries that relates a remote user name to the name of the distribution service unit at which the user can receive distributions.

From the perspective of the network, the directory is the collection of the individual user directories from all the DSUs in the network. Every distribution user must have at least one entry in the collective user directory.

The directory may include additional user information not related to the distribution service. The directory may also include entries for users not associated with the distribution service.

User Name. A two-part name for distribution users. It is intended to be a convenient and reasonably friendly name for users, e.g., ACCT.JONES or MKT.SMITH. It is not intended to include any reference to the user's location in the network.

Index

Printed in U.S.A.

Reader's Comment Form

Systems Network Architecture
Distribution Services
Reference

Publication No. SC30-3098-3

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: Copies of IBM Publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Possible topics for comment are: clarity, accuracy, completeness, organization, coding, retrieval, and legibility.

Comments:

What is your occupation?

If you wish a reply, give your name, company, mailing address, and date:

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SC30-3098-3

Reader's Comment Form

Fold and tape | **Please Do Not Staple** | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Dept. E74
P.O. Box 12195
Research Triangle Park, N.C. 27709-9990

Fold and tape | **Please Do Not Staple** | Fold and tape

IBM®

Reader's Comment Form

Systems Network Architecture
Distribution Services
Reference

Publication No. SC30-3098-3

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: Copies of IBM Publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Possible topics for comment are: clarity, accuracy, completeness, organization, coding, retrieval, and legibility.

Comments:

What is your occupation?

If you wish a reply, give your name, company, mailing address, and date:

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SC30-3098-3

Reader's Comment Form

Fold and tape **Please Do Not Staple** **Fold and tape**

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Dept. E74
P.O. Box 12195
Research Triangle Park, N.C. 27709-9990

Fold and tape **Please Do Not Staple** **Fold and tape**

IBM ®

Publication Number
SC30-3098-3

Printed in USA

SC30-3098-3