

WAIT_FOR_COMPLETION

Waits for posting to occur on one or more nonblocking operations represented in the specified list of wait objects. Posting of a nonblocking operation occurs when the LU has completed the associated nonblocking verb (or the operating system has completed a nonconversation operation represented in the list) and filled all the return values.

WAIT_FOR_COMPLETION	Option No. [113]
Supplied Parameters: WAIT_OBJECT_LIST (<i>variable</i>) TIMEOUT (<i>BLOCKING</i>) (<i>VALUE (variable)</i>)	
Returned Parameters: RETURN_CODE (<i>variable</i>) STATUS_LIST (<i>variable</i>)	

Supplied Parameters:

WAIT_OBJECT_LIST specifies the variable containing a list of the wait objects of the nonblocking operations for which posting is expected. Each wait object is represented in the list by a variable used to represent it in a previously issued nonblocking verb or nonconversation operation.

TIMEOUT specifies the variable containing the amount of time that the TP is willing to wait.

- **BLOCKING** specifies this is a blocking verb operation; i.e., the TP chooses to wait until a wait object is posted.
- **VALUE(variable)** specifies the maximum period of time (in microsecond units) the user is willing to wait.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- **OK** indicates that at least one wait object specified in **WAIT_OBJECT_LIST** has been posted.

WAIT_FOR_COMPLETION

- **ALL_UNKNOWN_WAIT_OBJECTS** indicates that none of the wait objects represented in **WAIT_OBJECT_LIST** is identifiable.
- **INVALID_TIMEOUT_VALUE** indicates that the **TIMEOUT** parameter contains a value that exceeds the range permitted by the implementation.
- **TIMER_EXPIRED** indicates that the specified timeout value has elapsed and none of the wait objects specified in **WAIT_OBJECT_LIST** has been posted.

STATUS_LIST specifies the variable that contains, on the completion of the verb, a list of the statuses corresponding to the wait objects represented in **WAIT_OBJECT_LIST**. The possible statuses of a wait object are:

- **NOT_POSTED** indicates that the associated operation has not yet completed.
- **POSTED** indicates the associated operation has completed and the return values for the associated verb or nonconversation operation have been filled.
- **UNKNOWN** indicates that the content of the wait object is not identifiable, either because the variable representing it was never specified in a nonblocking verb (or nonconversation verb) or because the variable's content were tampered with by the TP (if not protected by the operating system).

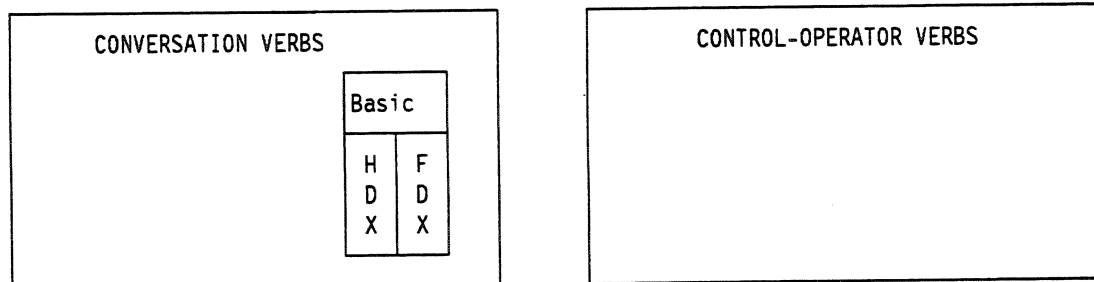
State Changes:

None

Notes:

1. The verb is intended to be implemented as a system function so that the TP can wait jointly for both conversation and nonconversation events, such as completions of file I/O, database queries, and interprocess communications. However, an implementation may choose to implement the verb only for the LU.
2. The **WAIT_OBJECT_LIST** parameter may specify wait objects for any combination of conversation and nonconversation events if the implementation permits.
3. The special timeout value 0 can be used to test all the wait objects represented in **WAIT_OBJECT_LIST**. The verb specified in this way incurs no blocking.
4. The verb returns an "OK" return code when at least one wait object in the list is posted, and returns "TIMER_EXPIRED" when none of the wait objects in the list is posted and timer expires. When all the wait objects in the list are unknown, it returns an "ALL_UNKNOWN_WAIT_OBJECTS" return code.

Chapter 7. Basic Conversation Verbs



This chapter describes the subcategory of conversation verbs called *basic conversation verbs*. These verbs are intended for use by LU services programs. The LU services programs can provide end-user services or protocol boundaries for end-user application transaction programs. Examples of LU services programs are:

- The LU services component programs that process mapped conversation verbs and control-operator verbs. These verbs define the LU 6.2 protocol boundary for mapped conversations and the control operator.
- SNA service transaction programs. These programs may provide end-user protocol boundaries that are defined by the specific IBM product implementations of the service programs. Refer to the IBM product publications for a description of the SNA service programs and their protocol boundaries that each product provides. The names of SNA service transaction programs are listed in Appendix D, "List of SNA Service Transaction Programs."

The detailed descriptions of the basic conversation verbs follow.

Note: Every conversation is either a basic or mapped conversation. The basic conversation verbs can be used for operations on both types. The mapped conversation verbs can be used for operations only on a mapped conversation. The capability to use basic conversation verbs on mapped conversations is provided for implementation of a mapped conversation LU services component program. Throughout the descriptions of the basic conversation verbs, references to a basic conversation or mapped conversation are made only when it is necessary to make a distinction between them. Otherwise, references are made simply to conversations.

ALLOCATE

ALLOCATE

Allocates a conversation between the local transaction program and a remote (partner) transaction program to a session between the local LU and the remote LU. This means that the local and remote programs have exclusive use of the session for the duration of the conversation. A resource ID is assigned to the conversation. This verb is issued prior to any verbs that refer to the conversation.

ALLOCATE	Option No. [Base]	Half-duplex [✓]	Full-duplex [✓]
Supplied Parameters:			
LU_NAME (variable) MODE_NAME (variable) TPN (variable)			
TYPE (<i>BASIC_CONVERSATION</i>)			
(<i>MAPPED_CONVERSATION</i>)			291
(<i>FULL_DUPLEX_BASIC_CONVERSATION</i>)			112
(<i>FULL_DUPLEX_MAPPED_CONVERSATION</i>)			112
RETURN_CONTROL			
(<i>WHEN_SESSION_ALLOCATED</i>)			
(<i>IMMEDIATE</i>)			203
(<i>WHEN_CONWINNER_ALLOCATED</i>)			201
(<i>WHEN_CONVERSATION_GROUP_ALLOCATED</i>)			401
(<i>WHEN_SESSION_FREE</i>)			205
CONVERSATION_GROUP_ID (variable)			401
SYNC_LEVEL			
(<i>NONE</i>)			
(<i>CONFIRM</i>)			
(<i>SYNCPT</i>)			108
SECURITY			
(<i>NONE</i>)			
(<i>SAME</i>)			212,217,219
(<i>PGM</i> (<i>USER_ID</i> (variable))			213
(<i>PASSWORD</i> (variable))			213
(<i>PROFILE</i> (variable))			218
PIP (<i>NO</i>)			
(<i>YES</i> (<i>variable1 variable2 ... variableN</i>))			241
WAIT_OBJECT			
(<i>BLOCKING</i>)			
(<i>VALUE</i> (variable))			113
Returned Parameters:			
RETURN_CODE (variable) RESOURCE (variable)			

Supplied Parameters:

LU_NAME specifies the name of the remote LU at which the remote transaction program is located. This LU name is any name by which the local LU knows the remote LU for the purpose of allocating a conversation. The local LU transforms this locally-known LU name to an LU name used by the network, if the names are different. See options 606 and 607 for locally known and uninterpreted LU names.

- Variable specifies the name of the LU (local or remote) where the partner transaction is located. (See Option# 204 for the special *intra-LU* case where the local TP and the partner TP are located at the same LU.)

MODE_NAME specifies the mode name designating the network properties for the session to be allocated for the conversation. The network properties include, for example, the class of service to be used, and whether data is to be enciphered before it is sent. The SNA-defined mode names, SNASVCMG or CPSVCMG, may be specified, but only by an LU services program. See "DEFINE_MODE" on page 10-9 for more information on the mode name.

TPN specifies the name of the remote transaction program to be connected at the other end of the conversation. A transaction program that has the appropriate privilege may specify the name of an SNA service transaction program. Privilege is an identification that a product or installation defines in order to differentiate LU services transaction programs from other programs, such as application transaction programs. (See Appendix D, "List of SNA Service Transaction Programs" for more details about SNA service transaction program names.)

TYPE specifies the type of conversation to be allocated.

- **BASIC_CONVERSATION** specifies to allocate a basic conversation.
- **MAPPED_CONVERSATION** specifies to allocate a mapped conversation. This argument is used in support of mapped conversation verbs. It may be specified only by a mapped conversation LU services program.
- **FULL_DUPLEX_BASIC_CONVERSATION** specifies to allocate a full-duplex basic conversation.
- **FULL_DUPLEX_MAPPED_CONVERSATION** specifies to allocate a full-duplex mapped conversation. This argument is used in support of mapped conversation verbs. It may be specified only by a mapped conversation LU services program.

RETURN_CONTROL specifies when the local LU is to return control to the local program, in relation to the allocation of a session for the conversation. An allocation error resulting from the local LU's failure to obtain a session for the conversation is reported on this verb. An allocation error resulting from the remote LU's rejection of the allocation request is reported on a subsequent verb.

ALLOCATE

- **WHEN_SESSION_ALLOCATED** specifies to allocate a session for the conversation before returning control to the program.
- **IMMEDIATE** specifies to allocate a session for the conversation if a session is immediately available. A session is immediately available when it is active, it is not allocated to another conversation, and the local LU is the contention winner for the session. If a session cannot be activated, then control reverts to the program with a return code of **UNSUCCESSFUL**.
- **WHEN_CONWINNER_ALLOCATED** specifies to allocate a contention-winner session for the conversation before returning control to the program.
- **WHEN_CONVERSATION_GROUP_ALLOCATED** specifies to allocate the session having the specified conversation group ID for the conversation before returning control to the program.
 - The **LU_NAME** parameter is omitted, it is implied.
 - The **MODE_NAME** parameter is omitted, it is implied.
 - The **CONVERSATION_GROUP_ID** parameter must be specified.
- **WHEN_SESSION_FREE** specifies to allocate a session for the conversation if a session is available. If no session is available, the local program is willing to wait for session activation. If a session cannot be activated and there are no outstanding session activation requests to satisfy this **ALLOCATE**, then control returns to the program with a return code of **ALLOCATION_FAILURE_RETRY**.

The following table contrasts all of the above **RETURN_CONTROL** parameters.

Table 7-1. RETURN_CONTROL Parameters and Conditions for Return of Control to the (MC_)ALLOCATE Verb			
	Con-Winner Session Acceptable?	Con-Loser Session Acceptable?	Wait Expected?
WHEN_SESSION_ALLOCATED	Yes	Yes	Yes, indefinitely
IMMEDIATE	Yes	No	No
WHEN_CONWINNER_ALLOCATED	Yes	No	Yes
WHEN_CONVERSATION_GROUP_ALLOCATED	A particular session is required		Yes, but only if conversation group is active
WHEN_SESSION_FREE	Yes	Yes	Yes, while there are outstanding session activation requests

CONVERSATION_GROUP_ID specifies the value of the conversation group ID that identifies a particular session when the **RETURN_CONTROL** parameter specifies **WHEN_CONVERSATION_GROUP_ALLOCATED**. When the **RETURN_CONTROL** parameter specifies a value other than **WHEN_CONVERSATION_GROUP_ALLOCATED**, this parameter is not used and is ignored.

The **CONVERSATION_GROUP_ID** is the identifier of a particular session between two particular LUs and provides a mechanism whereby one or more pairs of transaction programs (that is, one group of TPs at one LU and the other group of TPs at the other LU) can serially share the same identified session as if all the TPs were in a *conversation group* of intercommunicating TPs taking turns with their conversations over the same session. The conversation group ID is created by the LU and can be retrieved with the **GET_ATTRIBUTES** verb.

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

- **NONE** confirmation or sync-point processing on this conversation. The programs will not issue any verbs relating to these synchronization functions.
- **CONFIRM** specifies that the programs can perform confirmation processing but not sync-point processing on this conversation. The programs may issue verbs relating to confirmation, but they will not issue any verbs relating to sync point.

SYNC_LEVEL(CONFIRM) is not supported for full-duplex conversations.

- **SYNCPT** specifies that the programs can perform both confirmation and sync-point processing on this conversation. The programs may issue verbs relating to confirmation or sync point. For sync-point processing, a conversation allocated with this synchronization level is a protected resource.

SECURITY specifies access security information that the remote LU uses to verify the identity of the end-user and validate access to the remote program and its resources. The access security information consists of a user ID, a password, and a profile.

- **NONE** specifies to omit access security information on this allocation request.
- **SAME** specifies to send access security information which will cause the remote LU to create an equivalent security environment. If the remote LU trusts the local LU to verify user IDs and passwords, then the user ID and profile (if present) currently associated with the executing program is used. The access security information from the allocation request that initiates execution of a local program is used to initially set the associated user ID and profile values. The user ID is indicated as being already verified. If the remote LU does not trust the local LU to verify user IDs and passwords, or the local program does not have a user ID and profile associated with it, then the local LU, in conjunction with a local security manager, can attempt to determine user ID, password and profile value to send to the remote LU. This determination is implementation specific. If values cannot be determined, then access security information is omitted on this allocation request.
- **PGM** specifies to use the access security information that the local program provides on this parameter. The local program provides the information by means of the following arguments:

ALLOCATE

Security PGM parameter notes:

1. The **USER_ID** and **PASSWORD** parameters must always be specified as a pair if either is to be specified.
2. Specifying a null value for any of the access security arguments is equivalent to omitting the argument.

- **USER_ID** specifies the variable containing the user ID. The remote LU uses this value and the password to verify the identity of the end-user making the allocation request. In addition, the remote LU may use the user ID for auditing or accounting purposes, or it may use the user ID, together with the profile (if present), to determine which remote programs the local program may access and which resources the remote program may access.
- **PASSWORD** specifies the variable containing the password. The remote LU uses this value and the user ID to verify the identity of the end-user making the allocation request.
- **PROFILE** specifies the variable containing the profile. The remote LU may use this value, in addition to or in place of the user ID, to determine which remote programs the local program may access, and which resources the remote program may access.

PIP specifies program initialization parameters for the remote program.

- **NO** specifies that PIP data is not present.
- **YES** specifies that PIP data is present.
- **variable1 variable2 ... variableN** contain the PIP data to be sent to the remote program. The PIP data consists of one or more subfields, each of which is specified by a separate variable; variables 1 through n correspond to subfields 1 through n. If a variable is omitted in the PIP parameter or it is of null value, the corresponding PIP subfield is made to be of 0 length. The number and sequence of PIP subfields must agree with the number and sequence of PIP variables specified on the remote program's PROC statement (see "Transaction Program Structure and Execution" for a description of transaction program structure).

WAIT_OBJECT specifies whether the verb should be processed in a blocking or nonblocking manner. The default value is a blocking verb operation. If a non-blocking verb operation is to be requested, then **WAIT_OBJECT(VALUE(variable))** must be specified.

- **BLOCKING** specifies this is a blocking verb operation.
- **VALUE(variable)** specifies a nonblocking verb operation of one of two types:
 - a **null** variable specifies to complete the verb operation but without notification.
 - a nonnull **variable** specifies to complete the verb operation but with notification. The variable represents a wait object. Its use here associates it with completion of this verb. It can be listed as an entry in the **WAIT_OBJECT_LIST** in a subsequent **WAIT_FOR_COMPLETION** verb for notification (posting) purposes. Origination of this variable (by the TP or the operating system) is implementation dependent.

Returned Parameters:

RESOURCE specifies the variable in which the resource ID is to be returned. The length and actual format of the resource ID is product-dependent. The resource ID is returned to the program when the return code is either OK or ALLOCATION_ERROR.

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

The **RETURN_CONTROL** parameter affects the return codes that can be returned to the program. The following return codes can occur with all values of the **RETURN_CONTROL** parameter:

- OK
- ALLOCATION_ERROR, with one of the following subcodes (refer to Appendix F, "Conversation Return Codes" on page F-1 for a complete list of subcodes with their explanations):
 - SYNC_LEVEL_NOT_SUPPORTED_BY_LU
- OPERATION_INCOMPLETE
- OPERATION_NOT_ACCEPTED
- PROGRAM_PARAMETER_CHECK
 - MODE_NAME specifies SNASVCMG or CPSVCMG and the local program is not an LU services program.
 - SYNCPT is specified but is not supported.
 - TPN specifies an SNA service transaction program name and the local program does not have the appropriate privilege to allocate a conversation to an SNA service program.
 - TPN specifies a null (0 length) value.
 - TYPE(BASIC_CONVERSATION) is specified but the local program does not have basic conversation support defined.
 - TYPE(MAPPED_CONVERSATION) is specified but the local program is not a mapped conversation LU services program.
 - The RETURN_CONTROL value specified is not supported.
 - USER_ID and PASSWORD are not specified together.

Additional return codes are possible, depending on the **RETURN_CONTROL** value:

- RETURN_CONTROL(WHEN_SESSION_ALLOCATED) or RETURN_CONTROL(WHEN_CONWINNER_ALLOCATED):
 - ALLOCATION_ERROR, with one of the following subcodes (refer to Appendix F, "Conversation Return Codes" on page F-1 for a complete list of subcodes with their explanations):
 - ALLOCATION_FAILURE_NO_RETRY
 - ALLOCATION_FAILURE_RETRY
 - PARAMETER_ERROR, for one of the following reasons:
 - Invalid LU name
 - Invalid mode name
- RETURN_CONTROL(IMMEDIATE)
 - PARAMETER_ERROR, for one of the following reasons:
 - Invalid LU name

ALLOCATE

- Invalid mode name
 - UNSUCCESSFUL, for the following reason:
 - Session not immediately available
- RETURN_CONTROL(WHEN_CONVERSATION_GROUP_ALLOCATED)
 - ALLOCATION_ERROR, with one of the following subcodes (refer to Appendix F, "Conversation Return Codes" on page F-1 for a complete list of subcodes with their explanations):
 - ALLOCATION_FAILURE_NO_RETRY
- RETURN_CONTROL(WHEN_SESSION_FREE)
 - ALLOCATION_ERROR, with one of the following subcodes (refer to Appendix F, "Conversation Return Codes" on page F-1 for a complete list of subcodes with their explanations):
 - ALLOCATION_FAILURE_NO_RETRY
 - ALLOCATION_FAILURE_RETRY

Return codes for half-duplex conversations only:

The RETURN_CONTROL parameter affects the return codes that can be returned to the program. The following return codes can occur with all values of the RETURN_CONTROL parameter:

- PROGRAM_STATE_CHECK
 - New protected conversations may not be allocated when the program is in the backout-required state on other conversations.

Return codes for full-duplex conversations only:

- ALLOCATION_ERROR, with the following subcode (refer to Appendix F, "Conversation Return Codes" on page F-1 for a complete list of subcodes with their explanations):
 - FULL_DUPLEX_NOT_SUPPORTED_BY_LU
- PROGRAM_PARAMETER_CHECK, for one of the following reasons:
 - SYNC_LEVEL(CONFIRM) is specified but is not supported for full-duplex conversations.
 - A full-duplex conversation is specified but is not supported.

State Changes:

Half-duplex conversations only:

Send state is entered.

Full-duplex conversations only:

Send-receive state is entered.

Notes:

1. This verb is used by a transaction program to allocate a basic conversation. It is also used by an LU services component program to allocate either a basic conversation or a mapped conversation, depending on the function that the component program provides. For example, a component program that processes control operator verbs uses this verb to allocate a basic conversation, and a component program that processes mapped conversation verbs uses this verb to allocate a mapped conversation.
2. Depending on the product, the LU may send the allocation request to the remote LU as soon as it allocates a session for the conversation. Alternatively, the LU may buffer the allocation request until it accumulates from the PIP parameter of this verb or from one or more subsequent SEND_DATA verbs a sufficient amount of information for transmission, or until the local program issues a subsequent verb other than SEND_DATA that explicitly causes the LU to flush its send buffer. The amount of information that is sufficient for transmission depends on the characteristics of the session allocated for the conversation, and can vary from one session to another.
3. The local program can ensure that the remote program is connected as soon as possible by issuing FLUSH immediately after ALLOCATE.
4. Two LUs connected by a session may both attempt to allocate a conversation on the session at the same time. This is called contention. Contention is resolved by making one LU the contention winner of the session and the other LU the contention loser of the session. The contention-winner LU allocates a conversation on a session without asking permission from the contention-loser LU. Conversely, the contention-loser LU requests permission from the contention-winner LU to allocate a conversation on the session, and the contention-winner LU either grants or rejects the request.
5. If the program issues ALLOCATE with the parameter RETURN_CONTROL(WHEN_CONVERSATION_GROUP_ALLOCATED), and no active session exists for the conversation group ID specified on the CONVERSATION_GROUP_ID parameter, a return code of ALLOCATION_FAILURE_NO_RETRY is returned to the program. This error can occur, for example, when the conversation group was active at the time the GET_ATTRIBUTES verb was issued, but before the LU could allocate the conversation on the session the session ended (say, because of a session outage). This error can also occur when the conversation group ID is invalid.
6. Each LU indicates at session activation time whether it will accept LU security parameters on allocation requests the partner LU sends. If the remote LU will not accept any security parameters from the local LU, and the local program specifies SECURITY(SAME) or SECURITY(PGM(...)), the local LU downgrades the specification to SECURITY(NONE). Similarly, if the remote LU will not accept the local LU's verification of the user ID and password, and the local program specifies SECURITY(SAME), the local LU may attempt to determine user ID, profile and password values to send to the remote LU. If the local LU is unable to determine access security information to send it will downgrade the specification to SECURITY(NONE). When the remote LU does accept security parameters, the TP at the remote LU can obtain the user

ALLOCATE

ID and profile received in the allocation request by issuing the GET_TP_PROPERTIES verb.

7. The program uses the resource ID, returned to the program on the RESOURCE parameter, on all subsequent basic conversation verbs it issues for this conversation.
8. For programs that use multiple conversations, a request for RETURN_CONTROL(IMMEDIATE) or RETURN_CONTROL(WHEN_SESSION_FREE) should be used; the remaining RETURN_CONTROL parameters may cause the conversation allocation request to be queued indefinitely.
9. References in this verb description to a program being in a particular state are only in terms of the allocated conversation.

Half-duplex conversations only:

1. SYNC_LEVEL(SYNCPT) permits use of the SYNCPT and BACKOUT verbs to aid in maintaining consistency across all protected resources within a distributed logical unit of work. See *SNA LU 6.2 Reference: Peer Protocols* for more details of sync point.
2. Upon successful completion of the ALLOCATE, the local program is in send state and the remote program is in receive state.

Full-duplex conversations only:

1. Both the local and remote programs are connected in send-receive state.

CONFIRM

Sends a request for confirmation to a remote transaction program and waits for a reply. This verb allows the local and remote programs to interactively communicate during a particular time period for distributed processing between the two programs. The LU flushes its send buffer as a function of this verb.

CONFIRM	<i>Option No.</i> [Base]	<i>Half-duplex</i> [√]	<i>Full-duplex</i> []
Supplied Parameters:			
RESOURCE (<i>variable</i>)			
WAIT_OBJECT (<i>BLOCKING</i>) (<i>VALUE</i> (<i>variable</i>))			113
Returned Parameters:			
RETURN_CODE (<i>variable</i>)			
REQUEST_TO_SEND_RECEIVED (<i>variable</i>)			
EXPEDITED_DATA_RECEIVED (<i>variable</i>)			112

Supplied Parameters:

RESOURCE specifies the variable containing the resource ID. The conversation must be allocated with a synchronization level of CONFIRM or SYNCPT.

WAIT_OBJECT specifies whether the verb should be processed in a blocking or nonblocking manner. The default value is a blocking verb operation. If a non-blocking verb operation is to be requested, then **WAIT_OBJECT(VALUE(variable))** must be specified.

- **BLOCKING** specifies this is a blocking verb operation.
- **VALUE(variable)** specifies a nonblocking verb operation of one of two types:
 - a *null* variable specifies to complete the verb operation but without notification.
 - a nonnull *variable* specifies to complete the verb operation but with notification. The variable represents a wait object. Its use here associates it with completion of this verb. It can be listed as an entry in the **WAIT_OBJECT_LIST** in a subsequent **WAIT_FOR_COMPLETION** verb for notification (posting) purposes. Origination of this variable (by the TP or the operating system) is implementation dependent.

CONFIRM

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- OK (remote program replied CONFIRMED)
- ALLOCATION_ERROR
- BACKED_OUT
- DEALLOCATE_ABEND_PROG
- DEALLOCATE_ABEND_SVC
- DEALLOCATE_ABEND_TIMER
- OPERATION_INCOMPLETE
- OPERATION_NOT_ACCEPTED
- PROG_ERROR_PURGING
- PROGRAM_PARAMETER_CHECK, for one of the following reasons:
 - The conversation was allocated with SYNC_LEVEL(NONE).
 - RESOURCE specifies an unassigned resource ID.
 - This verb is not supported for full-duplex conversations.
- PROGRAM_STATE_CHECK, for one of the following reasons:
 - The conversation is not in send or defer-receive state. See Note 2 on page 7-13.
 - The conversation is in send state, and the program started but did not finish sending a logical record.
- RESOURCE_FAILURE_NO_RETRY
- RESOURCE_FAILURE_RETRY
- SVC_ERROR_PURGING

REQUEST_TO_SEND_RECEIVED specifies the variable in which is returned an indication of whether REQUEST_TO_SEND has been received.

- YES indicates a REQUEST_TO_SEND notification has been received from the remote transaction program. The remote program has issued REQUEST_TO_SEND, requesting the local program to enter receive state and thereby place the remote program in send state.
- NO indicates a REQUEST_TO_SEND notification has not been received.

EXPEDITED_DATA_RECEIVED specifies the variable in which is returned an indication of whether SEND_EXPEDITED_DATA has been received.

- YES indicates a SEND_EXPEDITED_DATA notification has been received from the remote transaction program.
- NO indicates a SEND_EXPEDITED_DATA notification has not been received from the remote transaction program.

State Changes:

Receive state is entered when the verb is issued in defer-receive state following PREPARE_TO_RECEIVE.

No state change occurs when the verb is issued in send state.

Notes:

1. The program may use this verb for various application-level functions. For example:
 - The program may issue this verb immediately following an `ALLOCATE` in order to determine whether the allocation of the conversation is successful before sending any data.
 - The program may issue this verb as a request for acknowledgement of data it sent to the remote program. The remote program may respond by issuing `CONFIRMED` as an indication that it received and processed the data without error, or by issuing `SEND_ERROR` as an indication that it encountered an error.
2. This verb cannot be issued in defer-deallocate state, a state entered when `DEALLOCATE TYPE(SYNC_LEVEL)` is issued when the synchronization level for the conversation is `SYNCPT`.
3. When `REQUEST_TO_SEND_RECEIVED` indicates YES, the remote program requests the local program to enter receive state and thereby place the remote program in send state. A program enters receive state by means of the `PREPARE_TO_RECEIVE` or `RECEIVE_AND_WAIT` verb. The partner program enters the corresponding send state when it issues a `RECEIVE_AND_WAIT` or `RECEIVE_IMMEDIATE` verb and receives the `SEND` indication (on the `WHAT_RECEIVED` parameter).
4. No state change has occurred because `EXPEDITED_DATA_RECEIVED` indicates YES.

When expedited data is received by the LU, it is indicated on the pertinent verbs if the return parameter `EXPEDITED_DATA_RECEIVED` exists on the verbs. It will continue to be indicated until such time that the data is received by the TP issuing the `RECEIVE_EXPEDITED_DATA` verb.
5. If a session becomes unavailable just prior to the issuance of the `CONFIRMED` verb (in response to the `CONFIRM` verb from the partner program), then the program issuing the `CONFIRMED` verb will not receive an indication of a failure until another verb is issued following the `CONFIRMED` verb.
6. References in this verb description to a program being in a particular state are only in terms of the specified conversation.

CONFIRMED

CONFIRMED

Sends a confirmation reply to the remote transaction program. This verb allows the local and remote programs to interactively communicate during a particular time period for distributed processing between the two programs. The local program can issue this verb when it receives a confirmation request (see the **WHAT_RECEIVED** parameter of the **RECEIVE_AND_WAIT** or **RECEIVE_IMMEDIATE** verb).

CONFIRMED	Option No. [Base]	Half-duplex [√]	Full-duplex []
Supplied Parameters:			
RESOURCE (<i>variable</i>)			
WAIT_OBJECT (BLOCKING) (<i>VALUE (variable)</i>)			
113			
Returned Parameters:			
RETURN_CODE (<i>variable</i>)			

Supplied Parameters:

RESOURCE specifies the variable containing the resource ID of the conversation.

WAIT_OBJECT specifies whether the verb should be processed in a blocking or nonblocking manner. The default value is a blocking verb operation. If a non-blocking verb operation is to be requested, then

WAIT_OBJECT(VALUE(variable)) must be specified.

- **BLOCKING** specifies this is a blocking verb operation.
- **VALUE(variable)** specifies a nonblocking verb operation of one of two types:
 - a *null* variable specifies to complete the verb operation but without notification.
 - a nonnull *variable* specifies to complete the verb operation but with notification. The variable represents a wait object. Its use here associates it with completion of this verb. It can be listed as an entry in the **WAIT_OBJECT_LIST** in a subsequent **WAIT_FOR_COMPLETION** verb for notification (posting) purposes. Origination of this variable (by the TP or the operating system) is implementation dependent.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- OK
- OPERATION_INCOMPLETE
- OPERATION_NOT_ACCEPTED
- PROGRAM_PARAMETER_CHECK, for one of the following reasons:
 - RESOURCE specifies an unassigned resource ID.
 - This verb is not supported for full-duplex conversations.
- PROGRAM_STATE_CHECK, for the following reason:
 - The conversation is not in confirm, confirm-send, or confirm-deallocate state.

State Changes:

Receive state is entered when CONFIRM was received on the preceding RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE.

Send state is entered when CONFIRM_SEND was received on the preceding RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE.

Deallocate state is entered when CONFIRM_DEALLOCATE was received on the preceding RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE.

Notes:

1. The program can issue this verb only as a reply to a confirmation request; the verb cannot be issued at any other time.
2. The program may use this verb for various application-level functions. For example, the remote program may send data followed by a confirmation request. When the local program receives the confirmation request, it may issue this verb as an indication that it received and processed the data without error, or by issuing SEND_ERROR as an indication that it encountered an error.
3. If a session becomes unavailable just prior to the issuance of the CONFIRMED verb (in response to the CONFIRM verb from the partner program), then the program issuing the CONFIRMED verb will not receive an indication of a failure until yet another verb is issued following the CONFIRMED verb.
4. References in this verb description to a program being in a particular state are only in terms of the specified conversation.

DEALLOCATE

DEALLOCATE

For half-duplex conversations:

Deallocates the specified conversation from the transaction program. The deallocation can be either completed as part of this verb, or deferred until the program issues a SYNCPT verb. When it is completed as part of this verb, it can include the function of the FLUSH or CONFIRM verb.

For full-duplex conversations:

DEALLOCATE with TYPE(FLUSH) closes the local program's send queue. Both the local and remote program must close their send queues independently; therefore, two DEALLOCATE TYPE(FLUSH) verbs are required to end the conversation. Notification that the partner has closed its send queue is given to the receive queue in the form of a DEALLOCATE_NORMAL return code.

DEALLOCATE with TYPE(ABEND) is an abrupt termination that will close both sides of the conversation simultaneously. This notification is returned to the remote program's send queue as an ERROR_INDICATION return code, and to remote program's receive queue as a DEALLOCATE_ABEND return code.

DEALLOCATE	Option No. [Base]	Half-duplex [√]	Full-duplex [√]
Supplied Parameters:			
RESOURCE (variable)			
TYPE(SYNC_LEVEL)			
(FLUSH)			
(CONFIRM)			
(ABEND_PROG)			
(ABEND_SVC)			
(ABEND_TIMER)		291	
(LOCAL)		291	
LOG_DATA			
(NO)			
(YES (variable))		290	
WAIT_OBJECT			
(BLOCKING)			
(VALUE (variable))		113	
Returned Parameters:			
RETURN_CODE (variable)			
EXPEDITED_DATA_RECEIVED (variable)		112	

Supplied Parameters:

RESOURCE specifies the variable containing the resource ID of the conversation to be deallocated.

TYPE specifies the type of deallocation to be performed.

- **SYNC_LEVEL** based on the synchronization level allocated to this conversation:
 - If **SYNC_LEVEL(NONE)**, execute the function of the **FLUSH** verb and deallocate the conversation normally.
 - If **SYNC_LEVEL(CONFIRM)**, execute the function of the **CONFIRM** verb and if it is successful (as indicated by a return code of OK on this **DEALLOCATE** verb), deallocate the conversation normally; if it is not successful, the state of the conversation is determined by the return code.
 - If **SYNC_LEVEL(SYNCPT)**, defer the deallocation until the program issues a **SYNCPT** verb. If the **SYNCPT** is successful (as indicated by a return code of OK on that verb), the conversation is then deallocated normally; otherwise, the state of the conversation is determined by the return code. See the **SYNCPT** verb for further discussion of state changes. If the program issues **BACKOUT**, the conversation is not deallocated.

- **FLUSH** specifies to execute the function of the **FLUSH** verb and deallocate the conversation normally.

TYPE(FLUSH) is not permitted on conversations allocated with **SYNC_LEVEL(SYNCPT)**.

- **CONFIRM** specifies to execute the function of the **CONFIRM** verb and if it is successful (as indicated by a return code of OK on this **DEALLOCATE** verb), deallocate the conversation normally; if it is not successful, the state of the conversation is determined by the return code.

TYPE(CONFIRM) is not permitted on conversations allocated with **SYNC_LEVEL(SYNCPT)**, and is not supported for full-duplex conversations.

- **ABEND_PROG**, **ABEND_SVC**, or **ABEND_TIMER** specifies to execute the function of the **FLUSH** verb when the conversation is in send or defer state, and deallocate the conversation abnormally. Logical-record truncation can occur when the conversation is in send state; data purging can occur when it is in receive state.
- **LOCAL** specifies to deallocate the conversation locally. This type of deallocation must be specified if, and only if, the conversation is in deallocate state. Deallocate state is entered when the program receives on a previously issued verb a return code indicating the conversation has been deallocated (see Appendix F, "Conversation Return Codes" and Note 5 on page 7-23).

TYPE(LOCAL) is not supported for full-duplex conversations.

The execution of the **FLUSH** or **CONFIRM** function as part of this verb includes the flushing of the LU's send buffer. When, instead, the deallocation is deferred, the LU also defers flushing its send buffer until the program issues a subsequent verb for this conversation.

DEALLOCATE

LOG_DATA specifies whether product-unique error information is to be placed in the system error logs of the LUs supporting this conversation. This parameter can be specified only when **TYPE(ABEND_PROG)**, **TYPE(ABEND_SVC)**, or **TYPE(ABEND_TIMER)** is also specified. Otherwise it is ignored.

- **NO** specifies that no error information is to be placed in the system error logs.
- **YES** specifies that product-unique error information is to be placed in the system error logs of the local and remote LUs. The specified variable contains the product-unique error information.

WAIT_OBJECT specifies whether the verb should be processed in a blocking or nonblocking manner. The default value is a blocking verb operation. If a non-blocking verb operation is to be requested, then **WAIT_OBJECT(VALUE(variable))** must be specified.

- **BLOCKING** specifies this is a blocking verb operation.
- **VALUE(variable)** specifies a nonblocking verb operation of one of two types:
 - a *null* variable specifies to complete the verb operation but without notification.
 - a nonnull *variable* specifies to complete the verb operation but with notification. The variable represents a wait object. Its use here associates it with completion of this verb. It can be listed as an entry in the **WAIT_OBJECT_LIST** in a subsequent **WAIT_FOR_COMPLETION** verb for notification (posting) purposes. Origination of this variable (by the TP or the operating system) is implementation dependent.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program. The **TYPE** parameter determines which of the following return codes can be returned to the program.

- A flush-level deallocate can be issued in two ways: with **TYPE(SYNC_LEVEL)** specified when the synchronization level allocated to this conversation is **NONE** or with **TYPE(FLUSH)** specified when the synchronization level allocated to this conversation is either **NONE** or **CONFIRM**. If a flush-level deallocate is issued, one of the following return codes is returned:
 - **OK** (deallocation is complete)
 - **OPERATION_INCOMPLETE**
 - **OPERATION_NOT_ACCEPTED**
 - **PROGRAM_PARAMETER_CHECK**, for one of the following reasons:
 - **RESOURCE** specifies an unassigned resource ID.
 - **PROGRAM_STATE_CHECK**, for one of the following reasons:
 - The conversation is in send or send/receive state, and the program started but did not finish sending a logical record.
- If **TYPE(CONFIRM)** is specified and the synchronization level allocated to this conversation is **NONE**, (for full-duplex conversations a synchronization level of **CONFIRM** is not allowed) one of the following return codes is returned:
 - **PROGRAM_PARAMETER_CHECK**, for one of the following reasons:

DEALLOCATE

- RESOURCE specifies an unassigned resource ID.
 - TYPE(CONFIRM) is specified and the conversation is allocated with SYNC_LEVEL(NONE).
- If TYPE(ABEND_PROG), TYPE(ABEND_SVC), or TYPE(ABEND_TIMER) is specified, one of the following return codes is returned:
 - OK (deallocation is complete)
 - OPERATION_INCOMPLETE
 - OPERATION_NOT_ACCEPTED
 - PROGRAM_PARAMETER_CHECK, for one of the following reasons:
 - RESOURCE specifies an unassigned resource ID.

Return codes for half-duplex conversations only:

- A confirm-level deallocate can be issued in two ways: with TYPE(SYNC_LEVEL) or with TYPE(CONFIRM) specified. In both cases, the synchronization level allocated to this conversation must be CONFIRM. If a confirm-level deallocate is issued, one of the following return codes is returned:
 - OK (deallocation is complete)
 - ALLOCATION_ERROR
 - DEALLOCATE_ABEND_PROG
 - DEALLOCATE_ABEND_SVC
 - DEALLOCATE_ABEND_TIMER
 - OPERATION_INCOMPLETE
 - OPERATION_NOT_ACCEPTED
 - PROG_ERROR_PURGING
 - PROGRAM_PARAMETER_CHECK, for the following reason:
 - RESOURCE specifies an unassigned resource ID.
 - PROGRAM_STATE_CHECK, for one of the following reasons:
 - The conversation is not in send state.
 - The conversation is in send state, and the program started but did not finish sending a logical record.
 - RESOURCE_FAILURE_NO_RETRY
 - RESOURCE_FAILURE_RETRY
 - SVC_ERROR_PURGING
- If TYPE(SYNC_LEVEL) is specified and the synchronization level allocated to this conversation is SYNCPT, one of the following return codes is returned:
 - OK (deallocation is deferred)
 - OPERATION_INCOMPLETE
 - OPERATION_NOT_ACCEPTED
 - PROGRAM_PARAMETER_CHECK, for the following reason:
 - RESOURCE specifies an unassigned resource ID.
 - PROGRAM_STATE_CHECK, for one of the following reasons:
 - The conversation is not in send state.
 - The conversation is in send state, and the program started but did not finish sending a logical record.

DEALLOCATE

- If TYPE(CONFIRM) or TYPE(FLUSH) is specified and the synchronization level allocated to this conversation is SYNCPT, the following return code is returned:
 - PROGRAM_PARAMETER_CHECK, for one of the following reasons:
 - RESOURCE specifies an unassigned resource ID.
 - TYPE(CONFIRM) or TYPE(FLUSH) is specified and the conversation is allocated with SYNC_LEVEL(SYNCPT).
- If TYPE(ABEND_PROG), TYPE(ABEND_SVC), or TYPE(ABEND_TIMER) is specified, one of the following return codes is returned:
 - PROGRAM_STATE_CHECK, for the following reason:
 - The conversation is not in send, defer, receive, confirm, or sync-point state.
- If TYPE(LOCAL) is specified, one of the following return codes is returned:
 - OK (deallocation is complete)
 - PROGRAM_PARAMETER_CHECK, for the following reason:
 - RESOURCE specifies an unassigned resource ID.
 - PROGRAM_STATE_CHECK, for the following reason:
 - The conversation is not in deallocate state.

Return codes for full-duplex conversations only:

- A flush-level deallocate can be issued in two ways: with either TYPE(SYNC_LEVEL) or TYPE(FLUSH) specified. If a flush-level deallocate is issued, one of the following return codes is returned:
 - ERROR_INDICATION (with one of the following subcodes) [Refer to Appendix F, "Conversation Return Codes" on page F-1 for a complete list of subcodes with their explanations.]
 - ALLOCATION_ERROR_PENDING
 - DEALLOCATE_ABEND_PROG_PENDING
 - DEALLOCATE_ABEND_SVC_PENDING
 - DEALLOCATE_ABEND_TIMER_PENDING
 - RESOURCE_FAILURE_NO_RETRY_PENDING
 - RESOURCE_FAILURE_RETRY_PENDING
 - UNKNOWN_ERROR_TYPE_PENDING
 - PROGRAM_STATE_CHECK, for the following reason:
 - The conversation is not in send-receive or send-only state.
- If TYPE(CONFIRM) is specified, the following return code is returned:
 - PROGRAM_PARAMETER_CHECK, for the following reason:
 - TYPE(CONFIRM) is not supported for full-duplex conversations.
- If TYPE(SYNC_LEVEL) is specified and the synchronization level allocated to the conversation is SYNCPT, the following return code is returned:
 - PROGRAM_PARAMETER_CHECK, for the following reason:
 - SYNC_LEVEL(SYNCPT) is not supported for full-duplex conversations.

DEALLOCATE

- If TYPE(LOCAL) is specified, the following return code is returned:
 - PROGRAM_PARAMETER_CHECK, for the following reason:
 - TYPE(LOCAL) is not supported for full-duplex conversations.

EXPEDITED_DATA_RECEIVED specifies the variable in which is returned an indication of whether SEND_EXPEDITED_DATA has been received.

- YES indicates a SEND_EXPEDITED_DATA notification has been received from the remote transaction program.
- NO indicates a SEND_EXPEDITED_DATA notification has not been received from the remote transaction program.

The EXPEDITED_DATA_RECEIVED variable is not set when the DEALLOCATE causes the conversation state to change to reset. The variable is set only if all of the following conditions hold:

- The conversation type (specified via the TYPE parameter on the ALLOCATE verb) is either FULL_DUPLEX_MAPPED_CONVERSATION or FULL_DUPLEX_BASIC_CONVERSATION.
- The DEALLOCATE TYPE specified is either FLUSH or SYNC_LEVEL.
- The DEALLOCATE verb is issued from send-receive state.

State Changes:

Half-duplex conversations only:

Deallocate-Pending state is entered when TYPE(CONFIRM) is specified and the reply to the CONFIRM from the remote program has not been received. This is a transient state. When the verb completes, the conversation enters **Reset** or **Receive** state.

Defer deallocate state is entered when TYPE(SYNC_LEVEL) is specified and the synchronization level is SYNCPT.

Reset state is entered when TYPE(FLUSH), TYPE(CONFIRM), TYPE(LOCAL), TYPE(ABEND_PROG), TYPE(ABEND_SVC), or TYPE(ABEND_TIMER) is specified; or when TYPE(SYNC_LEVEL) is specified and the synchronization level is NONE or CONFIRM.

Full-duplex conversations only:

Receive-only state is entered when a legitimate flush-level DEALLOCATE is specified and the current conversation state is send-receive.

Reset state is entered when either a legitimate flush-level DEALLOCATE is specified and the current conversation state is send-only, or when a TYPE(ABEND_PROG), TYPE(ABEND_SVC), or TYPE(ABEND_TIMER) is specified.

Notes:

DEALLOCATE

1. The TYPE(ABEND_PROG), TYPE(ABEND_SVC), and TYPE(ABEND_TIMER) parameters are intended to be used in order to unconditionally deallocate the conversation regardless of its synchronization level and its current state. Specifically:
 - The TYPE(ABEND_PROG) parameter is intended to be used by a transaction program when it detects an error condition that prevents further useful communications, that is, communications that would lead to successful completion of the transaction. The specific use and meaning of ABEND_PROG are program-defined.
 - The TYPE(ABEND_SVC) parameter is intended to be used by an LU services component, such as one that processes conversation verbs, when it detects an error condition caused by its peer LU services component in the remote LU. An example is a format error in control information sent by the peer LU services component. The specific use and meaning of ABEND_SVC are product-defined.
 - The TYPE(ABEND_TIMER) parameter is intended to be used by an LU services component, such as one that processes conversation verbs, when it detects or is informed of a condition that requires the conversation to be deallocated without further communications. For example, too much time elapses without receiving any information, or an operator prematurely ends program execution. The specific conditions and the means by which the LU services component detects or is informed of the conditions are product-defined. The specific use and meaning of ABEND_TIMER are also product-defined.
2. The EXPEDITED_DATA_RECEIVED indicator is not set when the conversation is in reset state.

Half-duplex conversations only:

1. When defer-deallocate state is entered, the LU buffers the deallocation information to be sent to the remote LU until the local program issues a SYNCPT verb that causes the LU to flush its send buffer. If the program issues the BACKOUT verb instead, the data is flushed but the deallocate information is not sent.
2. The TYPE(SYNC_LEVEL) parameter is intended to be used by the transaction program in order to deallocate the conversation based on the synchronization level allocated to the conversation.
 - If the synchronization level is NONE, the conversation is unconditionally deallocated.
 - If the synchronization level is CONFIRM, the conversation is deallocated when the remote program responds to the confirmation request by issuing CONFIRMED. The conversation remains allocated when the remote program responds to the confirmation request by issuing SEND_ERROR.
 - If the synchronization level is SYNCPT, the conversation is deallocated when the local program subsequently issues SYNCPT and receives an OK return code, indicating that all programs throughout the transaction responded to the sync-point request by issuing SYNCPT. The conversation remains allocated when the program receives a BACKED_OUT return code

to the SYNCPT verb indicating that a remote program responded to the sync-point request by issuing SEND_ERROR, or one or more programs responded by issuing BACKOUT.

3. The TYPE(FLUSH) parameter is intended to be used by the transaction program in order to unconditionally deallocate the conversation regardless of its synchronization level. TYPE(FLUSH) is functionally equivalent to TYPE(SYNC_LEVEL) with a synchronization level of NONE.

TYPE(FLUSH) is not legal on a conversation with SYNC_LEVEL(SYNCPT).

4. The TYPE(CONFIRM) parameter is intended to be used by the transaction program in order to conditionally deallocate the conversation, depending on the remote program's response, when the synchronization level is CONFIRM. TYPE(CONFIRM) is functionally equivalent to TYPE(SYNC_LEVEL) with a synchronization level of CONFIRM.

When a DEALLOCATE TYPE(CONFIRM) verb is issued in a nonblocking manner, its execution may be suspended because of waiting for a reply to the CONFIRM request from the remote program. If so suspended, the verb receives an OPERATION_INCOMPLETE return code, and the conversation enters deallocate-pending state. In this state, the program cannot exchange any expedited data.

The conversation is deallocated when the remote program responds to the confirmation request by issuing CONFIRMED. The conversation remains allocated when the remote program responds to the confirmation request by issuing SEND_ERROR.

TYPE(CONFIRM) is not legal on a conversation with SYNC_LEVEL(SYNCPT).

When DEALLOCATE TYPE(ABEND_*) is issued on a conversation with synchronization level SYNCPT, all protected conversations used by that program enter the backout-required state.

5. The TYPE(LOCAL) parameter is intended to be used by the transaction program in order to complete the program's deallocation of the conversation after receiving an indication that the conversation has been deallocated from the session, an indication such as a DEALLOCATE_NORMAL or RESOURCE_FAILURE_* return code.

A TYPE(LOCAL) deallocation of the conversation is necessary in order for the local LU to recognize that the local program no longer needs a conversation after the remote program has deallocated the conversation. It is analagous to a called party hanging up a telephone after the calling party has already done so.

6. The remote transaction program receives the deallocate notification by means of a return code or what-received indication, as follows:
 - DEALLOCATE_NORMAL return code: The local program specified either TYPE(FLUSH) or TYPE(SYNC_LEVEL) and the synchronization level is NONE.
 - CONFIRM_DEALLOCATE what-received indication: The local program specified either TYPE(CONFIRM) or TYPE(SYNC_LEVEL) and the synchronization level is CONFIRM.

DEALLOCATE

- TAKE_SYNCPT_DEALLOCATE what-received indication: The local program specified TYPE(SYNC_LEVEL), the synchronization level is SYNCPT, and the local program subsequently issued SYNCPT.
 - DEALLOCATE_ABEND_PROG, DEALLOCATE_ABEND_SVC, or DEALLOCATE_ABEND_TIMER return code: The local program specified, respectively, TYPE(ABEND_PROG), TYPE(ABEND_SVC), or TYPE(ABEND_TIMER), with the following exception: If the remote program has issued a SEND_ERROR in receive state, a DEALLOCATE_NORMAL return code is reported instead of one of the DEALLOCATE_ABEND return codes.
7. DEALLOCATE with TYPE(ABEND_PROG), TYPE(ABEND_SVC), or TYPE(ABEND_TIMER) resets or cancels posting. If posting is active and the conversation has been posted, posting is reset. If posting is active and the conversation has not been posted, posting is canceled (posting will not occur). See the POST_ON_RECEIPT verb for more details about posting.
 8. References in this verb description to a program being in a particular state are only in terms of the specified conversation.

Full-duplex conversations only:

1. DEALLOCATE TYPE(FLUSH) closes the local program's send queue. If a send queue verb is issued after the send queue has been closed, a PROGRAM_STATE_CHECK is returned. Both the local and remote program must close their send queues independently; therefore, two DEALLOCATE TYPE(FLUSH) verbs are required to end the conversation. The send queues are closed independently so that data in transit in both directions is delivered before ending the conversation.

When the DEALLOCATE TYPE(FLUSH) is issued from send-receive state, the conversation state changes to receive-only, and the program may continue to receive data until a DEALLOCATE_NORMAL return code is received.

2. When a transaction program receives a DEALLOCATE_NORMAL return code while in send-receive state, the conversation state transitions to send-only, and the receive queue is closed. The program may continue to send data until it is ready to end the conversation by issuing a DEALLOCATE TYPE(FLUSH).
3. An ERROR_INDICATION return code is returned when a terminating error has been received from the remote program; the DEALLOCATE TYPE(FLUSH) request is not performed. The terminating error may be a DEALLOCATE_ABEND, an ALLOCATION_ERROR, or a conversation failure. The nature of the terminating error is reflected in a subcode to the ERROR_INDICATION return code. If the nature of the terminating error is not known, a subcode of UNKNOWN_ERROR_TYPE_PENDING is returned.

When an ERROR_INDICATION return code is returned while the conversation state is send-only, the conversation state will change to reset. If an ERROR_INDICATION is returned while the conversation state is send-receive, no state change occurs; however, the send queue is effectively closed and subsequent send queue verbs will be rejected with the ERROR_INDICATION return code.

When an `ERROR_INDICATION` is returned while the conversation state is send-receive, the local program may end the conversation by either issuing receive queue verbs until the return code for the terminating error is received, or by issuing a `MC_DEALLOCATE` with `TYPE(ABEND)`.

4. When the subcode associated with the `ERROR_INDICATION` return code is `ALLOCATION_ERROR`, no additional information is returned regarding the nature of the `ALLOCATION_ERROR`.
5. `DEALLOCATE TYPE(ABEND_*)` is an abrupt termination that will close both sides of the conversation. Between the time when the local program's send queue begins processing the `DEALLOCATE TYPE(ABEND_*)` and that processing is completed, verbs outstanding on both the local and remote conversation queues may receive notification that a `DEALLOCATE TYPE(ABEND_*)` verb is in progress. The return codes used to give `DEALLOCATE TYPE(ABEND_*)` notification to the conversation queues are shown in Table 7-2.

When the `DEALLOCATE TYPE(ABEND_*)` verb completes, the conversation state changes to reset and any remaining outstanding verbs are posted with a `PROGRAM_PARAMETER_CHECK` return code.

Table 7-2. Notification That a `DEALLOCATE TYPE(ABEND_*)` Is in Progress.

	Local Program	Remote Program
send queue	<code>DEALLOCATE TYPE(ABEND_*)</code> is processed on this queue.	<code>ERROR_INDICATION (DEALLOCATE_ABEND_*_PENDING)</code>
receive queue	Data available to receive is returned, followed by <code>DEALLOCATE_ABEND_*</code> .	Data available to receive is returned, followed by <code>DEALLOCATE_ABEND_*</code> .
expedited-send queue	<code>CONVERSATION_ENDED</code>	<code>CONVERSATION_ENDED</code>
expedited-receive queue	Expedited data available to receive is returned, followed by <code>CONVERSATION_ENDED</code> .	Expedited data available to receive is returned, followed by <code>CONVERSATION_ENDED</code> .

6. An indication of `EXPEDITED_DATA_RECEIVED` may be returned on a `DEALLOCATE TYPE(FLUSH)` that was issued from send-receive state.
7. `DEALLOCATE` with `TYPE(CONFIRM)` or `TYPE(LOCAL)` is not supported.

FLUSH

FLUSH

Flushes the local LU's send buffer. The LU sends any information it has buffered to the remote LU. Information the LU buffers can come from ALLOCATE, SEND_DATA, PREPARE_TO_RECEIVE (see Note 2 below), or SEND_ERROR. Refer to the descriptions of these verbs for details of the information the LU buffers and when buffering occurs.

FLUSH	Option No. [101]	Half-duplex [✓]	Full-duplex [✓]
Supplied Parameters:			
RESOURCE (<i>variable</i>)			
WAIT_OBJECT (BLOCKING) (VALUE (<i>variable</i>))			
113			
Returned Parameters:			
RETURN_CODE (<i>variable</i>)			

Supplied Parameters:

RESOURCE specifies the variable containing the resource ID.

WAIT_OBJECT specifies whether the verb should be processed in a blocking or nonblocking manner. The default value is a blocking verb operation. If a non-blocking verb operation is to be requested, then **WAIT_OBJECT(VALUE(variable))** must be specified.

- **BLOCKING** specifies this is a blocking verb operation.
- **VALUE(variable)** specifies a nonblocking verb operation of one of two types:
 - a *null* variable specifies to complete the verb operation but without notification.
 - a nonnull *variable* specifies to complete the verb operation but with notification. The variable represents a wait object. Its use here associates it with completion of this verb. It can be listed as an entry in the **WAIT_OBJECT_LIST** in a subsequent **WAIT_FOR_COMPLETION** verb for notification (posting) purposes. Origination of this variable (by the TP or the operating system) is implementation dependent.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- OK
- OPERATION_INCOMPLETE
- OPERATION_NOT_ACCEPTED
- PROGRAM_PARAMETER_CHECK, for the following reason:
 - RESOURCE specifies an unassigned resource ID.

Return codes for half-duplex conversations only:

- PROGRAM_STATE_CHECK, for the following reason:
 - The conversation is not in send or defer-receive state.

Return codes for full-duplex conversations only:

- ERROR_INDICATION, with one of the following subcodes (Refer to Appendix F, "Conversation Return Codes" on page F-1 for a complete list of subcodes with their explanations):
 - ALLOCATION_ERROR_PENDING
 - DEALLOCATE_ABEND_PROG_PENDING
 - DEALLOCATE_ABEND_SVC_PENDING
 - DEALLOCATE_ABEND_TIMER_PENDING
 - RESOURCE_FAILURE_NO_RETRY_PENDING
 - RESOURCE_FAILURE_RETRY_PENDING
 - UNKNOWN_ERROR_TYPE_PENDING
- PROGRAM_STATE_CHECK, for the following reason:
 - The full-duplex conversation is not in send-receive or send-only state.

State Changes:

Half-duplex conversations only:

Receive state is entered when the verb is issued in defer state following PREPARE_TO_RECEIVE.

No state change occurs when the verb is issued in send state.

Full-duplex conversations only:

Reset state is entered when an ERROR_INDICATION return code is returned while in send-only state.

FLUSH

Notes:

1. This verb is useful for optimization of processing between the local and remote programs. The LU normally buffers the data from consecutive SEND_DATAs until it has a sufficient amount for transmission. At that time it transmits the buffered data. However, the local program can issue FLUSH in order to cause the LU to transmit the buffered data. In this way, the local program can minimize the delay in the remote program's processing of the data.
2. The LU flushes its send buffer only when it has some information to transmit. If the LU has no information in its send buffer, nothing is transmitted to the remote LU.
3. References in this verb description to a program being in a particular state are only in terms of the specified conversation.

Half-duplex conversations only:

1. This verb cannot legally be issued in defer-deallocate state, a state entered when DEALLOCATE TYPE(SYNC_LEVEL) is issued when the synchronization level for the conversation is SYNCPT.
2. This verb can be issued after PREPARE_TO_RECEIVE with TYPE(SYNC_LEVEL) when the synchronization level for the conversation is SYNCPT. The effect to the remote program is the same as issuing PREPARE_TO_RECEIVE with TYPE(FLUSH). The conversation enters receive state at the completion of the FLUSH verb.

Full-duplex conversations only:

1. In a half-duplex conversation, issuing a FLUSH does not guarantee that the partner will actually receive the flushed data. In full-duplex conversations, FLUSH will flush the local LU's send buffer and post the partner's RECEIVE_AND_WAIT FILL(BUFFER) request that data has arrived.
2. A full-duplex transaction program might choose to issue a FLUSH after issuing a SEND_ERROR in order to cause the FMH-7 to be sent to the partner program immediately. This is especially true if the program intends to begin receiving data after issuing the SEND_ERROR.
3. An ERROR_INDICATION return code is returned when a terminating error has been received from the remote program; the FLUSH request is not performed. The terminating error may be a DEALLOCATE_ABEND, an ALLOCATION_ERROR, or a conversation failure. The nature of the terminating error is reflected in a subcode to the ERROR_INDICATION return code. If the nature of the terminating error is not known, a subcode of UNKNOWN_ERROR_TYPE_PENDING is returned.

When an ERROR_INDICATION return code is returned while the conversation state is send-only, the conversation state will change to reset. If an ERROR_INDICATION is returned while the conversation state is send-receive, no state change occurs; however, the send queue is effectively closed and subsequent send queue verbs will be rejected with the ERROR_INDICATION return code.

When an `ERROR_INDICATION` is returned while the conversation state is send-receive, the local program may end the conversation by either issuing receive queue verbs until the return code for the terminating error is received, or by issuing a `DEALLOCATE` with `TYPE(ABEND)`.

4. When the subcode associated with the `ERROR_INDICATION` return code is `ALLOCATION_ERROR`, no additional information is returned regarding the nature of the `ALLOCATION_ERROR`.

GET_ATTRIBUTES

GET_ATTRIBUTES

Returns information pertaining to the specified conversation.

Note: The MC_GET_ATTRIBUTES verb is not in the LU 6.2 Base; this verb is.

GET_ATTRIBUTES	Option No. [Base]	Half-duplex [√]	Full-duplex [√]
Supplied Parameters:			
RESOURCE (<i>variable</i>)			
Returned Parameters:			
RETURN_CODE (<i>variable</i>)			
PARTNER_LU_NAME (<i>variable</i>)			
PARTNER_FULLY_QUALIFIED_LU_NAME (<i>variable</i>)			
MODE_NAME (<i>variable</i>)			
SYNC_LEVEL (<i>variable</i>)			
CONVERSATION_STATE (<i>variable</i>)			108
CONVERSATION_CORRELATOR (<i>variable</i>)			243 or 251
SESSION_ID (<i>variable</i>)			251
CONVERSATION_GROUP_ID (<i>variable</i>)			401

Supplied Parameters:

RESOURCE specifies the variable containing the resource ID of the conversation of which the attributes are desired.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- OK
- **PROGRAM_PARAMETER_CHECK**, for one of the following reasons:
 - **RESOURCE** specifies an unassigned resource ID.
 - **CONVERSATION_STATE** is specified and not supported.
 - **CONVERSATION_CORRELATOR** is specified and not supported.
 - **CONVERSATION_GROUP_ID** is specified and not supported.
 - **SESSION_ID** is specified and not supported.

PARTNER_LU_NAME specifies the variable for returning the name of the LU at which the remote transaction program is located. This is a name by which the local LU knows the remote LU for the purpose of allocating a conversation. Refer to the description of the LU_NAME parameter of ALLOCATE for more details.

PARTNER_FULLY_QUALIFIED_LU_NAME specifies the variable for returning the fully qualified name of the LU at which the remote transaction program is located. See "FULLY_QUALIFIED_LU_NAME" in "DEFINE_MODE" on page 10-9. If the partner's fully qualified LU name is not known, a null value is returned.

MODE_NAME specifies the variable for returning the mode name for the session on which the conversation is allocated.

SYNC_LEVEL specifies the variable for returning the level of synchronization processing being used for the conversation. The synchronization levels are:

- NONE
- CONFIRM
- SYNCPT

CONVERSATION_STATE specifies the variable for returning the current conversation state of the specified conversation. Possible values of the conversation state are shown in Appendix E, "Conversation State Matrices." All states shown in this appendix can be returned except the reset state, since a conversation with a recognized resource ID is never in this state.

CONVERSATION_CORRELATOR specifies the variable for returning the conversation correlator. The conversation correlator is created and maintained by the LU. The conversation correlator is useful to identify a conversation instance and, unlike the RESOURCE parameter, can be known globally. The LU uses it during sync-point resynchronization. If no conversation correlator is used on the conversation, a null value is returned.

SESSION_ID specifies the variable for returning the identifier of the session being used by the conversation. This parameter value can be used in a DEACTIVATE_SESSION verb.

CONVERSATION_GROUP_ID specifies the variable for returning the conversation group identifier. The CONVERSATION_GROUP_ID is the identifier of a particular session between two particular LUs and provides a mechanism whereby one or more pairs of transaction programs (i.e., one TP at one LU and the other TP at the other LU) can serially share the same identified session as if all the TPs were in a *conversation group* of intercommunicating TPs taking turns with their conversations over the same session.

State Changes:

None

Notes:

1. The program issues this verb in order to obtain the attributes of the conversation, including the one by which the program was started.

GET_ATTRIBUTES

2. The LU creates the conversation correlator for its use during sync-point resynchronization. For sync-point resynchronization, the conversation correlator correlates the logical unit of work to the sync-point states associated with the current instance of the local program.

POST_ON_RECEIPT

Causes the LU to post the specified conversation when information is available for the program to receive. The information can be data, conversation status, or a request for confirmation or sync point. WAIT should be issued after POST_ON_RECEIPT in order to wait for posting to occur. Alternatively, TEST may be issued following POST_ON_RECEIPT in order to determine when posting has occurred.

POST_ON_RECEIPT	Option No. [103,104]	Half-duplex [√]	Full-duplex []
Supplied Parameters:			
RESOURCE (<i>variable</i>) LENGTH (<i>variable</i>) FILL (<i>LL</i>) (<i>BUFFER</i>)			
Returned Parameters:			
RETURN_CODE (<i>variable</i>)			

Supplied Parameters:

RESOURCE specifies the variable containing the resource ID.

LENGTH specifies the variable containing a length value, which is the maximum length of data that the program can receive. This parameter is used along with FILL to determine when to post the conversation for the receipt of data.

FILL specifies whether posting for data is to occur in terms of the logical-record format of the data.

- **LL** logical record is received, or when a part of a logical record is received that is at least equal in length to that specified on the LENGTH parameter, whichever occurs first.
- **BUFFER** specifies to post when data (independent of its logical-record format) is available that is at least equal in length to that specified by the LENGTH parameter, or when the end of data is available, whichever occurs first.

The specification and effect of FILL(LL) versus FILL(BUFFER) is relevant only at the time the verb is issued. The specification does not depend on past use, and has no bearing on subsequent use, of this parameter on any verbs to which it applies (POST_ON_RECEIPT, RECEIVE_IMMEDIATE, and RECEIVE_AND_WAIT).

POST_ON_RECEIPT

Posting also occurs independent of the FILL specification when information other than data is received, such as conversation status (a SEND, PROG_ERROR_TRUNC, or DEALLOCATE_NORMAL indication, for example), or a confirmation or sync-point request.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- OK
- PROGRAM_PARAMETER_CHECK, for one of the following reasons:
 - RESOURCE specifies an unassigned resource ID.
 - This verb is not supported for full-duplex conversations.
- PROGRAM_STATE_CHECK, for the following reason:
 - The conversation is not in receive state.

State Changes:

None

Notes:

1. This verb is intended to be used in conjunction with TEST or WAIT. The use of this verb and WAIT allows a program to perform synchronous receiving from multiple conversations, where the program issues this verb for each of the conversations and then issues WAIT (for each conversation) to wait until information is available to be received on the conversations. The use of this verb and TEST allows a program to continue its processing and test the conversations to determine when information is available to be received.
2. Posting occurs when the LU has any information that the program can receive, such as data, conversation status, or a request for confirmation or sync point. Refer to the RECEIVE_AND_WAIT verb for a description of the types of information a program can receive.
3. Posting is active for a conversation when POST_ON_RECEIPT has been issued for the conversation and posting has not yet been reset or canceled.

Posting is reset when any of the following verbs is issued for the same conversation as specified on POST_ON_RECEIPT *after* the conversation is posted:

- BACKOUT
- DEALLOCATE with TYPE(ABEND_PROG), TYPE(ABEND_SVC), or TYPE(ABEND_TIMER)
- RECEIVE_AND_WAIT
- RECEIVE_IMMEDIATE
- SEND_ERROR
- TEST
- WAIT

POST_ON_RECEIPT

Posting is canceled when any of the following verbs is issued for the same conversation as specified on POST_ON_RECEIPT *before* the conversation is posted:

- BACKOUT
- DEALLOCATE with TYPE(ABEND_PROG), TYPE(ABEND_SVC), or TYPE(ABEND_TIMER)
- RECEIVE_AND_WAIT
- RECEIVE_IMMEDIATE
- SEND_ERROR

In order for the program to activate posting again after posting has been reset or canceled, the program issues another POST_ON_RECEIPT.

4. Any number of POST_ON_RECEIPTs may be issued for a given conversation before posting is reset or canceled. The last POST_ON_RECEIPT issued for a conversation is the one that determines when posting will occur for data. For example, if a program issues POST_ON_RECEIPT with FILL(BUFFER) and LENGTH(1000) in preparation to receive 1000 bytes of data, and then issues the verb again with LENGTH(500), posting will occur when 500 bytes of data are available; or if the program issues the verb again with FILL(LL), posting will occur in terms of logical records.
5. POST_ON_RECEIPT with LENGTH(0) has no special significance. It specifies that posting for data is to occur upon receipt of any amount of data of one byte or more. It is equivalent to POST_ON_RECEIPT with LENGTH(1).
6. When FILL(BUFFER) is specified, posting for data occurs independent of its logical record format. The conversation is posted when an amount of data is available that is equal to, or less than, the length specified on the LENGTH parameter. Posting for less data can occur only when the end of the data is available. The end of data occurs when it is followed by an indication of a change in the state of the conversation, that is, a change to send, confirm, sync-point, or deallocate state. See RECEIVE_AND_WAIT for additional information.
7. References in this verb description to a program being in a particular state are only in terms of the specified conversation.

PREPARE_FOR_SYNCPT

PREPARE_FOR_SYNCPT

Causes the protected resources associated with the conversation to be prepared to advance to the next synchronization point. Protected resources are those currently allocated to the transaction with a synchronization level of SYNCPT. As part of the PREPARE_FOR_SYNCPT verb processing, the LU flushes its send buffer for the conversation.

PREPARE_FOR_SYNCPT	Option No. [111]	Half-duplex [√]	Full-duplex []
Supplied Parameters:			
RESOURCE (<i>variable</i>)			
WAIT_OBJECT (BLOCKING) (VALUE (<i>variable</i>))			
113			
Returned Parameters:			
RETURN_CODE (<i>variable</i>)			

Supplied Parameters:

RESOURCE specifies the variable containing the resource ID of the conversation to be prepared to commit.

WAIT_OBJECT specifies whether the verb should be processed in a blocking or nonblocking manner. The default value is a blocking verb operation. If a non-blocking verb operation is to be requested, then

WAIT_OBJECT(VALUE(variable)) must be specified.

- **BLOCKING** specifies this is a blocking verb operation.
- **VALUE(variable)** specifies a nonblocking verb operation of one of two types:
 - a **null** variable specifies to complete the verb operation but without notification.
 - a nonnull **variable** specifies to complete the verb operation but with notification. The variable represents a wait object. Its use here associates it with completion of this verb. It can be listed as an entry in the **WAIT_OBJECT_LIST** in a subsequent **WAIT_FOR_COMPLETION** verb for notification (posting) purposes. Origination of this variable (by the TP or the operating system) is implementation dependent.

Returned Parameters:

RETURN_CODE specifies the variable in which the return code is returned to the transaction program. The return code indicates whether the prepare operation failed, or caused the conversation partner to prepare to commit, or caused the conversation partner to backout.

- **OK:** Protected resources at the conversation partner are prepared to commit. All protected resources subordinate to the conversation partner are also prepared to commit.
- **BACKED_OUT:** Protected resources at the conversation partner have been backed out. The **BACKED_OUT** return code has one of the following sub-codes, which indicates whether the backout occurred consistently in the rest of the distributed transaction subordinate to the conversation partner.
 - **ALL_AGREED:** All protected resources in the distributed transaction have backed out.
 - **LUW_OUTCOME_PENDING:** As a result of failures and at least one program in the distributed transaction specifying **SET_SYNCPT_OPTIONS WAIT_FOR_OUTCOME(NO)**, it is not known whether all protected resources have successfully backed out.
 - **LUW_OUTCOME_MIXED:** At least one protected resource has been advanced to the next synchronization point as a result of operator intervention following a failure. The distributed transaction is damaged.
 - **OPERATION_INCOMPLETE**
 - **OPERATION_NOT_ACCEPTED**
- **PROGRAM_PARAMETER_CHECK**, for one of the following reasons:
 - **RESOURCE** specifies an unassigned resource ID.
 - The synchronization level of the conversation is not **SYNCPT**.
 - This verb is not supported for full-duplex conversations.
- **PROGRAM_STATE_CHECK**, for one of the following reasons:
 - The conversation is not in send, defer-receive, or defer-deallocate state.
 - The conversation is in send state, and the program started but did not finish sending a logical record.
- **ALLOCATION_ERROR**
- **DEALLOCATE_ABEND_PROG**
- **DEALLOCATE_ABEND_SVC**
- **DEALLOCATE_ABEND_TIMER**
- **PROG_ERROR_PURGING**
- **PROG_ERROR_NO_TRUNC**
- **SVC_ERROR_PURGING**
- **SVC_ERROR_NO_TRUNC**
- **RESOURCE_FAILURE_NO_RETRY**
- **RESOURCE_FAILURE_RETRY**

State Changes:

From send state the conversation enters the sync-point-send state.

From defer-receive state the conversation enters the sync-point state.

From defer-deallocate state the conversation enters the sync-point-deallocate state.

State Changes (when RETURN_CODE indicates BACKED_OUT):

PREPARE_FOR_SYNCPT

Backout-required state is entered.

Notes:

1. The program may issue PREPARE_FOR_SYNCPT on a protected conversation that is in send, defer-receive, or defer-deallocate state. The PREPARE_FOR_SYNCPT verb causes a sync point request to be sent on the conversation. The remote program receives the sync-point request by means of a TAKE_SYNCPT_* value of the WHAT_RECEIVED parameter of the RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb. When the remote program receives this indicator, it may issue one of the following, with the specified effect:
 - SYNCPT: causes the sync-point request to be propagated to its other protected resources.
 - BACKOUT: causes the backout operation to be propagated to its subordinate protected resources, and a back out message to be returned to the LU of the local program. The local program that issued the PREPARE_FOR_SYNCPT verb gets a BACKED_OUT return code. All other protected conversations at the local program enter the backout-required state.
 - DEALLOCATE TYPE(ABEND): causes all protected conversations at the local program to enter the backout-required state.
 - SEND_ERROR: The local program gets a *_ERROR_NO_TRUNC return code. The resulting conversation state for the local program is receive state.
2. The remote program may have issued a SEND_ERROR verb before receiving the TAKE_SYNCPT_* indication. If this occurs, control is returned to the local program with *_ERROR_PURGING return code and the conversation state changes to receive state.
3. If the local program receives an error return code that puts it in receive state, it can then attempt to fix the error and retry the sync-point operation. This is a major difference with the SYNCPT verb: if an error indication to data sent earlier is received during execution of a SYNCPT verb, the transaction is backed out without giving the program a chance to fix it.
4. RETURN_CODE indicates whether the conversation partner has successfully prepared to commit. The subcode to RETURN_CODE indicates the action taken by the programs in the distributed transaction that are subordinate to the conversation partner, i.e., the transaction programs to which it propagates the sync-point operation.
5. References in this verb description to a program being in a particular state are only in terms of each resource.

PREPARE_TO_RECEIVE

Changes the conversation from send to receive state in preparation to receive data. The change to receive state can be either completed as part of this verb, or deferred until the program issues a FLUSH, CONFIRM, or SYNCPT verb. When it is completed as part of this verb it includes the function of the FLUSH or CONFIRM verb.

PREPARE_TO_RECEIVE	Option No. [105]	Half-duplex [✓]	Full-duplex []
Supplied Parameters:			
RESOURCE (<i>variable</i>)			
TYPE (<i>SYNC_LEVEL</i>)			
(<i>FLUSH</i>)			
(<i>CONFIRM</i>)			
LOCKS			
(<i>SHORT</i>)			
(<i>LONG</i>)			
WAIT_OBJECT			
(<i>BLOCKING</i>)			
(<i>VALUE</i> (<i>variable</i>))			
Returned Parameters:			
RETURN_CODE (<i>variable</i>)			

Supplied Parameters:

- RESOURCE** specifies the variable containing the resource ID of the conversation.
- TYPE** specifies the type of prepare-to-receive to be performed for this conversation.
- **SYNC_LEVEL** specifies to perform the prepare-to-receive based on the synchronization level allocated to this conversation:
 - If **SYNC_LEVEL(NONE)**, execute the function of the FLUSH verb and enter receive state.
 - If **SYNC_LEVEL(CONFIRM)**, execute the function of the CONFIRM verb and if it is successful (as indicated by a return code of OK on this **PREPARE_TO_RECEIVE** verb), enter receive state; if it is not successful, the state of the conversation is determined by the return code.
 - If **SYNC_LEVEL(SYNCPT)**, enter defer state until the program issues a SYNCPT, or the program issues a CONFIRM or FLUSH for this conver-

PREPARE_TO_RECEIVE

sation. If the SYNCPT or CONFIRM is successful (as indicated by a return code of OK on that verb) or FLUSH is issued, receive state is then entered for this conversation; otherwise, the state of the conversation is determined by the return code.

- **FLUSH** specifies to execute the function of the FLUSH verb and enter receive state.
- **CONFIRM** specifies to execute the function of the CONFIRM verb and if it is successful (as indicated by a return code of OK on this PREPARE_TO_RECEIVE verb), enter receive state; if it is not successful, the state of the conversation is determined by the return code.

The execution of the FLUSH or CONFIRM function as part of this verb includes the flushing of the LU's send buffer. When, instead, defer state is entered, the LU defers flushing its send buffer until the program issues a subsequent verb for this conversation.

LOCKS specifies when control is to be returned to the local program following execution of the CONFIRM function of this verb or following execution of a CONFIRM verb issued subsequent to this verb. This parameter is significant only when TYPE(CONFIRM) is also specified, or when TYPE(SYNC_LEVEL) is also specified and the synchronization level for this conversation is CONFIRM; or when TYPE(SYNC_LEVEL) is also specified, the synchronization level for this conversation is SYNCPT, and a subsequent CONFIRM or SYNCPT is issued. Otherwise, this parameter has no meaning and is ignored.

- **SHORT** specifies to return control when an affirmative reply is received, as follows:
 - When the synchronization level is CONFIRM, return control from execution of this verb when a CONFIRMED reply is received.
 - When the synchronization level is SYNCPT, return control immediately from execution of this verb; return control from execution of a subsequent CONFIRM or SYNCPT verb when a corresponding CONFIRMED or SYNCPT reply is received.
- **LONG** specifies to return control when information, such as data, is received from the remote program following an affirmative reply, as follows:
 - When the synchronization level is CONFIRM, return control from execution of this verb when information is received following a CONFIRMED reply.
 - When the synchronization level is SYNCPT, return control immediately from execution of this verb; return control from execution of a subsequent CONFIRM or SYNCPT verb when information is received following a corresponding CONFIRMED or SYNCPT reply.

WAIT_OBJECT specifies whether the verb should be processed in a blocking or nonblocking manner. The default value is a blocking verb operation. If a non-blocking verb operation is to be requested, then WAIT_OBJECT(VALUE(variable)) must be specified.

- **BLOCKING** specifies this is a blocking verb operation.
- **VALUE(variable)** specifies a nonblocking verb operation of one of two types:

PREPARE_TO_RECEIVE

- a *null* variable specifies to complete the verb operation but without notification.
- a nonnull *variable* specifies to complete the verb operation but with notification. The variable represents a wait object. Its use here associates it with completion of this verb. It can be listed as an entry in the WAIT_OBJECT_LIST in a subsequent WAIT_FOR_COMPLETION verb for notification (posting) purposes. Origination of this variable (by the TP or the operating system) is implementation dependent.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program. The TYPE parameter and synchronization level of the conversation affect the return codes that can be returned to the program.

The following return codes can be returned for all values of the TYPE parameter with any synchronization level:

- OK
- OPERATION_INCOMPLETE
- OPERATION_NOT_ACCEPTED
- PROGRAM_PARAMETER_CHECK, for one of the following reasons:
 - RESOURCE specifies an unassigned resource ID.
 - The TYPE parameter is inconsistent with the synchronization level of the conversation, e.g., TYPE(CONFIRM) is specified and the conversation is allocated with SYNC_LEVEL(NONE).
 - This verb is not supported for full-duplex conversations.
- PROGRAM_STATE_CHECK, for one of the following reasons:
 - The conversation is not in send state.
 - The conversation is in send state, and the program started but did not finish sending a logical record.

Additional return codes are possible if TYPE(CONFIRM) is specified or if TYPE(SYNC_LEVEL) is specified when the synchronization level allocated to the conversation is CONFIRM:

- ALLOCATION_ERROR
- BACKED_OUT (can only occur when the synchronization level of the conversation is SYNCPT)
- DEALLOCATE_ABEND_PROG
- DEALLOCATE_ABEND_SVC
- DEALLOCATE_ABEND_TIMER
- PROG_ERROR_PURGING
- SVC_ERROR_PURGING
- RESOURCE_FAILURE_NO_RETRY
- RESOURCE_FAILURE_RETRY

State Changes:

Defer state is entered when TYPE(SYNC_LEVEL) is specified and the synchronization level is SYNCPT.

PREPARE_TO_RECEIVE

Receive state is entered when TYPE(FLUSH) or TYPE(CONFIRM) is specified, or when TYPE(SYNC_LEVEL) is specified and the synchronization level is NONE or CONFIRM.

Notes:

1. The TYPE(SYNC_LEVEL) parameter is intended to be used by the transaction program in order to transfer send control to the remote program based on the synchronization level allocated to the conversation.
 - If the synchronization level is NONE, send control is transferred to the remote program without any synchronizing acknowledgment.
 - If the synchronization level is CONFIRM, send control is transferred to the remote program with confirmation requested.
 - If the synchronization level is SYNCPT, transfer of send control is deferred. When the local program subsequently issues SYNCPT, FLUSH, or CONFIRM, send control is transferred to the remote program with sync point requested.
2. The TYPE(FLUSH) parameter is intended to be used by the transaction program in order to transfer send control to the remote program without any synchronizing acknowledgment. TYPE(FLUSH) is functionally equivalent to:
 - TYPE(SYNC_LEVEL) with a synchronization level of NONE.
 - TYPE(SYNC_LEVEL) with a synchronization level of SYNCPT, followed by the FLUSH verb.
3. The TYPE(CONFIRM) parameter is intended to be used by the transaction program in order to transfer send control to the remote program with confirmation requested. TYPE(CONFIRM) is functionally equivalent to:
 - TYPE(SYNC_LEVEL) with a synchronization level of CONFIRM.
 - TYPE(SYNC_LEVEL) with a synchronization level of SYNCPT, followed by the CONFIRM verb.
4. The remote transaction program receives send control by means of a what-received indication of SEND, CONFIRM_SEND, or TAKE_SYNCPT_SEND, as follows:
 - SEND: The local program specified either TYPE(FLUSH); TYPE(SYNC_LEVEL) and the synchronization level is NONE; or TYPE(SYNC_LEVEL), the synchronization level is SYNCPT, and the local program subsequently issued FLUSH.
 - CONFIRM_SEND: The local program specified either TYPE(CONFIRM); TYPE(SYNC_LEVEL) and the synchronization level is CONFIRM; or TYPE(SYNC_LEVEL), the synchronization level is SYNCPT, and the local program subsequently issued CONFIRM.
 - TAKE_SYNCPT_SEND: The local program specified TYPE(SYNC_LEVEL), the synchronization level is SYNCPT, and the local program subsequently issued SYNCPT.

PREPARE_TO_RECEIVE

5. If TYPE(SYNC_LEVEL) is specified and the synchronization level for the conversation is SYNCPT, the LU buffers the SEND notification to be sent to the remote program until the local program issues a verb that causes the LU to flush its send buffer.
6. The conversation for the remote program enters the corresponding send state when it issues a RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb and receives the SEND indication (on the WHAT_RECEIVED parameter.) The remote program can then send data to the local program.
7. The use of LOCKS(LONG) can result in a reduction of line flows at the cost of longer completion of confirm or sync-point processing.
8. References in this verb description to a program being in a particular state are only in terms of the specified conversation.

RECEIVE_AND_WAIT

RECEIVE_AND_WAIT

Waits for information to arrive on the specified conversation and then receives the information. If information is already available, the program receives it without waiting. The information can be data, conversation status, or a request for confirmation or sync point. Control is returned to the program with an indication of the type of information.

For half-duplex conversations:

The program can issue this verb when the conversation is in send state. In this case, the LU flushes its send buffer, sending all buffered information and the SEND indication to the remote program. This changes the conversation to receive state. The LU then waits for information to arrive. The remote program can send data to the local program after it receives the SEND indication.

For full-duplex conversations:

If the send buffer contains the conversation allocation request, it will be flushed; otherwise, this verb will not cause the LU to flush its send buffer. If it is important that the data remaining in the send buffer be transmitted before receiving data, the local program should issue a FLUSH before issuing this verb.

RECEIVE_AND_WAIT	Option No. [Base]	Half-duplex [✓]	Full-duplex [✓]
Supplied Parameters:			
RESOURCE (variable)			
FILL (LL) (BUFFER)			
WAIT_OBJECT (BLOCKING) (VALUE (variable))			
Supplied-and-Returned Parameters:			
LENGTH (variable)			
Returned Parameters:			
RETURN_CODE (variable)			
REQUEST_TO_SEND_RECEIVED (variable)			
EXPEDITED_DATA_RECEIVED (variable)			
DATA (variable)			
WHAT_RECEIVED (variable)			

Supplied Parameters:

RESOURCE specifies the variable containing the resource ID of the conversation.

FILL specifies whether the program is to receive data in terms of the logical-record format of the data.

- **LL** specifies the program is to receive one complete or truncated logical record, or a portion of the logical record that is equal to the length specified by the **LENGTH** parameter.
- **BUFFER** specifies the program is to receive data independent of its logical-record format. The amount of data received will be equal to, or less than, the length specified by the **LENGTH** parameter. The amount is less than the specified length when the program receives the end of the data.

The specification and effect of **FILL(LL)** versus **FILL(BUFFER)** is relevant only at the time the verb is issued. The specification does not depend on past use, and has no bearing on subsequent use, of this parameter on any verbs to which it applies (**POST_ON_RECEIPT**, **RECEIVE_AND_WAIT**, and **RECEIVE_IMMEDIATE**).

WAIT_OBJECT specifies whether the verb should be processed in a blocking or nonblocking manner. The default value is a blocking verb operation. If a non-blocking verb operation is to be requested, then **WAIT_OBJECT(VALUE(variable))** must be specified.

- **BLOCKING** specifies this is a blocking verb operation.
- **VALUE(variable)** specifies a nonblocking verb operation of one of two types:
 - a *null* variable specifies to complete the verb operation but without notification.
 - a nonnull *variable* specifies to complete the verb operation but with notification. The variable represents a wait object. Its use here associates it with completion of this verb. It can be listed as an entry in the **WAIT_OBJECT_LIST** in a subsequent **WAIT_FOR_COMPLETION** verb for notification (posting) purposes. Origination of this variable (by the TP or the operating system) is implementation dependent.

Supplied-and-Returned Parameters:

LENGTH specifies the variable containing a length value that is the maximum amount of data the program is to receive. When control is returned to the program this variable contains the actual amount of data the program received up to the maximum. If the program receives information other than data, this variable remains unchanged.

Returned Parameters:

RECEIVE_AND_WAIT

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program. The return codes that can be returned depend on the state of the conversation at the time this verb is issued. The verb may be issued in send state or receive state. For full-duplex conversations, the verb may be issued from either send-receive state or receive-only state. State changes for full-duplex conversations occur only if a **DEALLOCATE_*** or **ALLOCATION_ERROR** return code is received.

- OK
- **ALLOCATION_ERROR**
- **DEALLOCATE_ABEND_PROG**
- **DEALLOCATE_ABEND_SVC**
- **DEALLOCATE_ABEND_TIMER**
- **DEALLOCATE_NORMAL**
- **OPERATION_INCOMPLETE**
- **OPERATION_NOT_ACCEPTED**
- **PROG_ERROR_NO_TRUNC**
- **PROG_ERROR_TRUNC** (cannot occur in send state)
- **PROGRAM_PARAMETER_CHECK**, for the following reason:
 - **RESOURCE** specifies an unassigned resource ID.
- **RESOURCE_FAILURE_NO_RETRY**
- **RESOURCE_FAILURE_RETRY**
- **SVC_ERROR_NO_TRUNC**
- **SVC_ERROR_TRUNC** (cannot occur in send state)

Return codes for half-duplex conversations only:

- **BACKED_OUT**
- **PROG_ERROR_PURGING**
- **PROGRAM_STATE_CHECK**, for one of the following reasons:
 - The conversation is not in send or receive state.
 - The conversation is in send state, and the program started but did not finish sending a logical record.
- **SVC_ERROR_PURGING** (cannot occur in reconnect state)

Return codes for full-duplex conversations only:

- **PROGRAM_STATE_CHECK**, for the following reason:
 - The full-duplex conversation is not in send/receive or receive-only state.

REQUEST_TO_SEND_RECEIVED specifies the variable in which is returned an indication of whether **REQUEST_TO_SEND** has been received.

- YES indicates a **REQUEST_TO_SEND** notification has been received from the remote transaction program. The remote program has issued **REQUEST_TO_SEND**, requesting the local program to enter receive state and thereby place the remote program in send state.
- NO indicates a **REQUEST_TO_SEND** notification has not been received.

This parameter is always set to NO for full-duplex conversations.

EXPEDITED_DATA_RECEIVED specifies the variable in which is returned an indication of whether **SEND_EXPEDITED_DATA** has been received. NO.

- YES indicates a **SEND_EXPEDITED_DATA** notification has been received from the remote transaction program.

- NO indicates a SEND_EXPEDITED_DATA notification has not been received from the remote transaction program.

DATA specifies the variable in which the program is to receive the data. When the program receives information other than data, as indicated by the **WHAT_RECEIVED** parameter, nothing is placed in this variable.

WHAT_RECEIVED specifies the variable in which is returned an indication of what the transaction program received. The program should examine this variable only when **RETURN_CODE** indicates OK; otherwise, nothing is placed in this variable.

- **DATA** is indicated when **FILL(BUFFER)** is specified and data (independent of its logical-record format) is received by the program.
- **DATA_COMPLETE** is indicated when **FILL(LL)** is specified and a complete logical record, or the last remaining portion thereof, is received by the program.
- **DATA_INCOMPLETE** is indicated when **FILL(LL)** is specified and less than a complete logical record is received by the program. The transaction program issues another **RECEIVE_AND_WAIT** (or possibly multiple **RECEIVE_AND_WAITs**) to receive the remainder of the data.
- **LL_TRUNCATED** is indicated when **FILL(LL)** is specified and the 2-byte **LL** field of a logical record has been truncated after the first byte. The LU discards the truncated **LL** field; it is not received by the program.

WHAT_RECEIVED for half-duplex conversations only:

- **SEND** indicates the remote program has entered receive state, placing the local program in send state. The local program may now issue **SEND_DATA**.
- **CONFIRM** indicates the remote program has issued **CONFIRM**, requesting the local program to respond by issuing **CONFIRMED**. The program may respond, instead, by issuing a verb other than **CONFIRMED**, such as **SEND_ERROR**.
- **CONFIRM_SEND** indicates the remote program has issued **PREPARE_TO_RECEIVE** with **TYPE(CONFIRM)**; or with **TYPE(SYNC_LEVEL)**, and either the synchronization level is **CONFIRM**, or it is **SYNCPT** and the remote program subsequently issued **CONFIRM**. The local program may respond by issuing **CONFIRMED**, or by issuing another verb such as **SEND_ERROR**.
- **CONFIRM_DEALLOCATE** indicates the remote program has issued **DEALLOCATE** with **TYPE(CONFIRM)**; or with **TYPE(SYNC_LEVEL)**, and either the synchronization level is **CONFIRM**, or it is **SYNCPT** and the remote program subsequently issued **CONFIRM**. The local program may respond by issuing **CONFIRMED**, or by issuing another verb such as **SEND_ERROR**.
- **TAKE_SYNCPT** indicates the remote program has issued **SYNCPT** or **PREPARE_FOR_SYNCPT**, requesting the local program to respond by issuing **SYNCPT** in order to perform the sync-point function on all protected resources throughout the transaction. Issuing the **SYNCPT** verb also causes an affirmative reply to be returned to the remote program if the sync-point function is successful. The program may respond, instead, by issuing a verb other than **SYNCPT**, such as **BACKOUT** or **SEND_ERROR**.

RECEIVE_AND_WAIT

- **TAKE_SYNCPT_SEND** indicates the remote program has issued **PREPARE_TO_RECEIVE** with **TYPE(SYNC_LEVEL)**, the synchronization level is **SYNCPT**, and the remote program subsequently issued either **SYNCPT** or **PREPARE_FOR_SYNCPT**. Issuing the **SYNCPT** verb also causes an affirmative reply to be returned to the remote program if the sync-point function is successful. The local program may respond by issuing **SYNCPT**, or by issuing another verb such as **BACKOUT** or **SEND_ERROR**.
- **TAKE_SYNCPT_DEALLOCATE** indicates the remote program has issued **DEALLOCATE** with **TYPE(SYNC_LEVEL)**, the synchronization level is **SYNCPT**, and the remote program subsequently issued either **SYNCPT** or **PREPARE_FOR_SYNCPT**. Issuing the **SYNCPT** verb also causes an affirmative reply to be returned to the remote program if the sync-point function is successful. The local program may respond by issuing **SYNCPT**, or by issuing another verb such as **BACKOUT** or **SEND_ERROR**.

State Changes:

Half-duplex conversations only:

Receive state is entered when the verb is issued in send state and **WHAT_RECEIVED** indicates **DATA**, **DATA_COMPLETE**, **DATA_INCOMPLETE**, or **LL_TRUNCATED**.

Send state is entered when **WHAT_RECEIVED** indicates **SEND**.

Confirm state is entered when **WHAT_RECEIVED** indicates **CONFIRM**, **CONFIRM_SEND**, or **CONFIRM_DEALLOCATE**.

Sync point state is entered when **WHAT_RECEIVED** indicates **TAKE_SYNCPT**, **TAKE_SYNCPT_SEND**, or **TAKE_SYNCPT_DEALLOCATE**.

No state change occurs when the verb is issued in receive state and **WHAT_RECEIVED** indicates **DATA**, **DATA_COMPLETE**, **DATA_INCOMPLETE**, or **LL_TRUNCATED**.

Full-duplex conversations only:

Reset state is entered when an **ALLOCATION_ERROR** return code is received, when a **DEALLOCATE_NORMAL** return code is received while in receive-only state, or a **DEALLOCATE_ABEND_*** return code is received.

Send-only state is entered when a **DEALLOCATE_NORMAL** return code is received while in send-receive state.

Notes:

1. When **FILL(LL)** is specified, the program is to receive a logical record and there are the following possibilities:
 - The program receives a complete logical record or the last remaining portion of a complete record. The length of the record or portion of the record is equal to or less than the length specified on the **LENGTH** parameter. The **WHAT_RECEIVED** parameter indicates **DATA_COMPLETE**.

RECEIVE_AND_WAIT

- The program receives an incomplete logical record. The logical record is incomplete because:
 - The length of the logical record is greater than the length specified on the LENGTH parameter; in this case the amount received equals the length specified.
 - Only a portion of the logical record is available because it has been truncated, the portion being equal to or less than the length specified on the LENGTH parameter.

The WHAT_RECEIVED parameter indicates DATA_INCOMPLETE. The program issues another RECEIVE_AND_WAIT (or possibly multiple RECEIVE_AND_WAITs) to receive the remainder of the logical record.

- The program receives no part of the logical record because it was truncated after the first byte of the LL field. The WHAT_RECEIVED parameter indicates LL_TRUNCATED.

Refer to the SEND_DATA verb for a definition of complete and incomplete logical records.

2. RECEIVE_AND_WAIT with LENGTH(0) has no special significance. The type of information available is indicated by the RETURN_CODE and WHAT_RECEIVED parameters, as usual. If data is available and FILL(LL) is specified, the WHAT_RECEIVED parameter indicates DATA_INCOMPLETE. If data is available and FILL(BUFFER) is specified, the WHAT_RECEIVED parameter indicates DATA. In either case, however, the program receives no data.
3. The program receives only one kind of information at a time. For example, it may receive data or a CONFIRM request, but it does not receive both at the same time. Also, if the remote program truncates a logical record, the local program receives the indication of the truncation on the RECEIVE_AND_WAIT it issues after receiving all of the truncated record. The RETURN_CODE and WHAT_RECEIVED parameters indicate to the program the kind of information the program receives.
4. No state change has occurred because EXPEDITED_DATA_RECEIVED indicates YES.

When expedited data is received by the LU, it is indicated on the pertinent verbs if the return parameter EXPEDITED_DATA_RECEIVED exists on the verbs. It will continue to be indicated until such time that the data is received by the TP issuing the RECEIVE_EXPEDITED_DATA verb.
5. References in this verb description to the program being in a particular state are only in terms of the specified conversation.

Half-duplex conversations only:

1. When the program issues RECEIVE_AND_WAIT in send state, the LU implicitly executes a PREPARE_TO_RECEIVE with TYPE(FLUSH) before executing the RECEIVE_AND_WAIT. Refer to the description of PREPARE_TO_RECEIVE for details of its function.

RECEIVE_AND_WAIT

2. When FILL(BUFFER) is specified, the program is to receive data independent of its logical-record format. The program receives an amount of data equal to, or less than, the length specified on the LENGTH parameter. The program can receive less data only when it receives the end of the data. The end of data occurs when it is followed by an indication of a change in the state of the conversation, that is, a change to send, confirm, sync-point, or deallocate state. The program is responsible for tracking the logical-record format of the data.
3. RECEIVE_AND_WAIT includes posting. If posting is already active when this verb is issued, this verb supersedes the prior POST_ON_RECEIPT function. Posting is reset at the completion of this verb. See the POST_ON_RECEIPT verb for more details about posting.
4. The REQUEST_TO_SEND notification is usually received when the local transaction program is in send state, and reported to the program on a SEND_DATA verb or on a SEND_ERROR verb issued in send state. However, the notification can be received when the program is in receive state under the following conditions:
 - When the local program just entered receive state and the remote program issued REQUEST_TO_SEND before it entered send state.
 - When the remote program has just entered receive state by means of the PREPARE_TO_RECEIVE verb (not RECEIVE_AND_WAIT), and then issued REQUEST_TO_SEND before the local program enters send state. This can occur because the REQUEST_TO_SEND is transmitted as an expedited request and can therefore arrive ahead of the request carrying the SEND indication. Potentially, the local program cannot distinguish this condition from the first. This ambiguity is avoided when the remote program waits until it receives information from the local program before it issues the REQUEST_TO_SEND.
 - When the remote program issues the REQUEST_TO_SEND in send state (see "Notes on Implementation Details" on page A-53).
5. The REQUEST_TO_SEND notification is returned to the program in addition to (not in place of) the information indicated by the RETURN_CODE and WHAT_RECEIVED parameters.

Full-duplex conversations only:

1. The REQUEST_TO_SEND notification is always set to NO.
2. When FILL(BUFFER) is specified, the program is to receive data independent of its logical-record format. The program receives an amount of data equal to, or less than, the length specified on the LENGTH parameter. The program can receive less data only when it receives the end of the data. The end of data indication for full-duplex conversations will be received only when the conversation has ended, the partner has issued a FLUSH, or an error has occurred. The program is responsible for tracking the logical-record format of the data.
3. If the conversation allocation request has been buffered and has not yet been sent to the partner, the RECEIVE_AND_WAIT processing will flush the send buffer and the allocation request will be sent.

RECEIVE_EXPEDITED_DATA

Receives data sent by the remote transaction program in an expedited manner, via the SEND_EXPEDITED_DATA verb. The program can issue this verb in any conversation state except for reset state.

RECEIVE_EXPEDITED_DATA	Option No. [112]	Half-duplex [√]	Full-duplex [√]
Supplied Parameters:			
RESOURCE (<i>variable</i>) RETURN_CONTROL (<i>WHEN_EXPEDITED_DATA_RECEIVED</i>) (<i>IMMEDIATE</i>) WAIT_OBJECT (<i>BLOCKING</i>) (<i>VALUE</i> (<i>variable</i>))			
113			
Supplied-and-Returned Parameters:			
LENGTH (<i>variable</i>)			
Returned Parameters:			
RETURN_CODE (<i>variable</i>) REQUEST_TO_SEND_RECEIVED (<i>variable</i>) EXPEDITED_DATA_RECEIVED (<i>variable</i>) DATA (<i>variable</i>)			
112			

Supplied Parameters:

RESOURCE specifies the variable containing the resource ID.

RETURN_CONTROL specifies when the local LU is to return control to the local program, in relation to the receiving of expedited data for the conversation.

- **WHEN_EXPEDITED_DATA_RECEIVED** specifies to receive the expedited data for the conversation before returning control to the program.
- **IMMEDIATE** specifies to receive the expedited data for the conversation if expedited data is immediately available and return control to the program with a return code indicating whether expedited data has been received.

RECEIVE_EXPEDITED_DATA

WAIT_OBJECT specifies whether the verb should be processed in a blocking or nonblocking manner. The default value is a blocking verb operation. If a non-blocking verb operation is to be requested, then

WAIT_OBJECT(VALUE(variable)) must be specified.

- **BLOCKING** specifies this is a blocking verb operation.
- **VALUE(variable)** specifies a nonblocking verb operation of one of two types:
 - a *null* variable specifies to complete the verb operation but without notification.
 - a nonnull *variable* specifies to complete the verb operation but with notification. The variable represents a wait object. Its use here associates it with completion of this verb. It can be listed as an entry in the **WAIT_OBJECT_LIST** in a subsequent **WAIT_FOR_COMPLETION** verb for notification (posting) purposes. Origination of this variable (by the TP or the operating system) is implementation dependent.

Supplied-and-Returned Parameters:

LENGTH specifies the variable containing a length value that is the buffer size the program has provided to receive the expedited data. If insufficient buffer size (including **LENGTH = 0**) is specified on the **LENGTH** parameter while the program is receiving the expedited data, a return code of **INSUFFICIENT_BUFFER_SIZE_PROVIDED** is returned to the program. In this case, the **LENGTH** value returned indicates the actual amount of data that is available and nothing is placed in the **DATA** parameter. If the program specifies a buffer size that is greater than or equal to the actual amount of data that is received, **LENGTH** is also set to the amount actually received. The valid expedited data size may vary from 0-86 bytes.

The program may issue the **RECEIVE_EXPEDITED_DATA** verb to test if there is expedited data to be received rather than receive the data to its buffer. This can be done by specifying **LENGTH = 0** and **RETURN_CONTROL = IMMEDIATE** in the **RECEIVE_EXPEDITED_DATA** verb. If expedited data is available, the return code **INSUFFICIENT_BUFFER_SIZE_PROVIDED** is returned to the program. The program can then decide what to do next. If expedited data is not available, a return code **UNSUCCESSFUL** is returned to the program and the **LENGTH** field is not significant.

Returned Parameters:

RETURN_CODE specifies the variable in which a return code is returned to the program. The return code indicates the result of verb execution. The verb may be issued in any state except in reset state.

- **OK**
- **CONVERSATION_ENDED**
- **EXPEDITED_DATA_NOT_SUPPORTED_BY_LU**
- **INSUFFICIENT_BUFFER_SIZE_PROVIDED**, for the following reason:

RECEIVE_EXPEDITED_DATA

- The expedited data the program received exceeds the receive buffer that the program was provided. The expedited data record cannot be partially received.
- OPERATION_INCOMPLETE
- OPERATION_NOT_ACCEPTED
- PROGRAM_PARAMETER_CHECK, for one of the following reasons:
 - RESOURCE specifies an unassigned resource ID
 - Expedited data is not supported
- PROGRAM_STATE_CHECK, for the following reason:
 - The conversation is in deallocate-pending state.

An additional return code is possible depending on the RETURN_CONTROL value:

- RETURN_CONTROL(IMMEDIATE)
 - UNSUCCESSFUL
 - Expedited data not immediately available

REQUEST_TO_SEND_RECEIVED specifies the variable in which is returned an indication of whether REQUEST_TO_SEND has been received.

- YES indicates a REQUEST_TO_SEND notification has been received from the remote transaction program.
- NO indicates a REQUEST_TO_SEND notification has not been received from the remote transaction program.

This parameter is always set to NO for full-duplex conversations only.

EXPEDITED_DATA_RECEIVED specifies the variable in which is returned an indication of whether SEND_EXPEDITED_DATA has been received.

- YES indicates a SEND_EXPEDITED_DATA notification has been received from the remote transaction program.
- NO indicates a SEND_EXPEDITED_DATA notification has not been received from the remote transaction program.

DATA specifies the variable in which the program is to receive the data. When the program provides insufficient buffer size, as indicated by the LENGTH parameter, nothing is placed in this variable.

State Changes:

None

Notes:

1. For RETURN_CONTROL(WHEN_EXPEDITED_DATA_RECEIVED), if a normal conversation deallocation occurs when a RECEIVE_EXPEDITED_DATA verb is outstanding, and no expedited data is available to receive, a return code of CONVERSATION_ENDED will be returned. CONVERSATION_ENDED is also returned when a DEALLOCATE(ABEND), allocation error, or conversation failure occurs. This return code indicates that a RECEIVE_EXPEDITED_DATA verb was

RECEIVE_EXPEDITED_DATA

outstanding when the conversation ended, the verb was not completed successfully, and the expedited-receive queue is effectively closed.

No state change occurs because CONVERSATION_ENDED is returned. Subsequent RECEIVE_EXPEDITED_DATA verbs will not be performed, but will be rejected with the CONVERSATION_ENDED return code.

2. If a SEND_EXPEDITED_DATA verb is issued from the local program, a RECEIVE_EXPEDITED_DATA verb needs to be issued by the remote program to receive the expedited data.
3. No state change has occurred because EXPEDITED_DATA_RECEIVED indicates YES.

When expedited data is received by the LU, it is indicated on the pertinent verbs if the return parameter EXPEDITED_DATA_RECEIVED exists on the verbs. It will continue to be indicated until such time that the data is received by the TP issuing the RECEIVE_EXPEDITED_DATA verb.

Full-duplex conversations only:

1. The REQUEST_TO_SEND notification is always set to NO.

RECEIVE_IMMEDIATE

Receives any information that is available from the specified conversation, but does not wait for information to arrive. The information (if any) can be data, conversation status, or a request for confirmation or sync point. Control is returned to the program with an indication of whether any information was received and, if so, the type of information.

When the send-receive queue for half-duplex conversations or the receive queue for full-duplex conversations is not empty, the verb can be blocked just like the RECEIVE_AND_WAIT verb.

RECEIVE_IMMEDIATE	<i>Option No.</i> [106]	<i>Half-duplex</i> [✓]	<i>Full-duplex</i> [✓]
Supplied Parameters:			
RESOURCE (<i>variable</i>)			
FILL (<i>LL</i>) (<i>BUFFER</i>)			
WAIT_OBJECT (<i>BLOCKING</i>) (<i>VALUE</i> (<i>variable</i>))			
Supplied-and-Returned Parameters:			
LENGTH (<i>variable</i>)			
Returned Parameters:			
RETURN_CODE (<i>variable</i>)			
REQUEST_TO_SEND_RECEIVED (<i>variable</i>)			
EXPEDITED_DATA_RECEIVED (<i>variable</i>)			
DATA (<i>variable</i>)			
WHAT_RECEIVED (<i>variable</i>)			

Supplied Parameters:

RESOURCE specifies the variable containing the resource ID of the conversation.

FILL specifies whether the program is to receive data in terms of the logical-record format of the data.

- **LL** specifies the program is to receive one logical record, or whatever portion of the logical record is available, up to the length specified by the **LENGTH** parameter.

RECEIVE_IMMEDIATE

- **BUFFER** specifies the program is to receive data independent of its logical-record format, up to the length specified by the **LENGTH** parameter.

The specification and effect of **FILL(LL)** versus **FILL(BUFFER)** is relevant only at the time the verb is issued. The specification does not depend on past use, and has no bearing on subsequent use, of this parameter on any verbs to which it applies (**POST_ON_RECEIPT**, **RECEIVE_IMMEDIATE**, and **RECEIVE_AND_WAIT**).

WAIT_OBJECT specifies whether the verb should be processed in a blocking or nonblocking manner. The default value is a blocking verb operation. If a non-blocking verb operation is to be requested, then **WAIT_OBJECT(VALUE(variable))** must be specified.

- **BLOCKING** specifies this is a blocking verb operation.
- **VALUE(variable)** specifies a nonblocking verb operation of one of two types:
 - a *null* variable specifies to complete the verb operation but without notification.
 - a nonnull *variable* specifies to complete the verb operation but with notification. The variable represents a wait object. Its use here associates it with completion of this verb. It can be listed as an entry in the **WAIT_OBJECT_LIST** in a subsequent **WAIT_FOR_COMPLETION** verb for notification (posting) purposes. Origination of this variable (by the TP or the operating system) is implementation dependent.

Supplied-and-Returned Parameters:

LENGTH specifies the variable containing a length value that is the maximum amount of data the program is to receive. When control is returned to the program this variable contains the actual amount of data the program received up to the maximum. If the program receives information other than data, or no information at all, this variable remains unchanged.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- **OK**
- **ALLOCATION_ERROR**
- **DEALLOCATE_ABEND_PROG**
- **DEALLOCATE_ABEND_SVC**
- **DEALLOCATE_ABEND_TIMER**
- **DEALLOCATE_NORMAL**
- **OPERATION_INCOMPLETE**
- **OPERATION_NOT_ACCEPTED**
- **PROG_ERROR_NO_TRUNC**
- **PROG_ERROR_TRUNC**
- **PROGRAM_PARAMETER_CHECK**, for the following reason:
 - **RESOURCE** specifies an unassigned resource ID.
- **RESOURCE_FAILURE_NO_RETRY**

RECEIVE_IMMEDIATE

- RESOURCE_FAILURE_RETRY
- SVC_ERROR_NO_TRUNC
- SVC_ERROR_TRUNC
- UNSUCCESSFUL - There is nothing to receive.

Return codes for half-duplex conversations only:

- BACKED_OUT
- PROG_ERROR_PURGING
- PROGRAM_STATE_CHECK, for the following reason:
 - The conversation is not in receive state.
- SVC_ERROR_PURGING

Return codes for full-duplex conversations only:

- PROGRAM_STATE_CHECK, for the following reason:
 - The full-duplex conversation is not in send-receive or receive-only state.

REQUEST_TO_SEND_RECEIVED specifies the variable in which is returned an indication of whether REQUEST_TO_SEND has been received.

- YES indicates a REQUEST_TO_SEND notification has been received from the remote program. The remote program has issued REQUEST_TO_SEND, requesting the local program to enter receive state and thereby place the remote program in send state.
- NO indicates a REQUEST_TO_SEND notification has not been received.

Note: If RETURN_CODE is set to PROGRAM_PARAMETER_CHECK or PROGRAM_STATE_CHECK, the value contained in REQUEST_TO_SEND_RECEIVED is meaningless.

This parameter is always set to NO for full-duplex conversations.

EXPEDITED_DATA_RECEIVED specifies the variable in which is returned an indication of whether SEND_EXPEDITED_DATA has been received.

- YES indicates a SEND_EXPEDITED_DATA notification has been received from the remote transaction program.
- NO indicates a SEND_EXPEDITED_DATA notification has not been received from the remote transaction program.

DATA specifies the variable in which the program is to receive the data. When the program receives information other than data, as indicated by the WHAT_RECEIVED parameter, nothing is placed in this variable.

WHAT_RECEIVED specifies the variable in which is returned an indication of what the transaction program received. The program should examine this variable only when RETURN_CODE indicates OK; otherwise, nothing is placed in this variable.

- DATA is indicated when FILL(BUFFER) is specified and data (independent of its logical-record format) is received by the program.
- DATA_COMPLETE is indicated when FILL(LL) is specified and a complete logical record, or the last remaining portion thereof, is received by the program.
- DATA_INCOMPLETE is indicated when FILL(LL) is specified and less than a complete logical record is received by the program. The transaction program

RECEIVE_IMMEDIATE

issues another RECEIVE_IMMEDIATE (or possibly multiple RECEIVE_IMMEDIATEs) to receive the remainder of the logical record.

- LL_TRUNCATED is indicated when FILL(LL) is specified and the 2-byte LL field of a logical record has been truncated after the first byte. The LU discards the truncated LL field; it is not received by the program.

WHAT_RECEIVED for half-duplex conversations only:

- SEND indicates the remote program has entered receive state, placing the local program in send state. The local program may now issue SEND_DATA.
- CONFIRM indicates the remote program has issued CONFIRM, requesting the local program to respond by issuing CONFIRMED. The program may respond, instead, by issuing a verb other than CONFIRMED, such as SEND_ERROR.
- CONFIRM_SEND indicates the remote program has issued PREPARE_TO_RECEIVE with TYPE(CONFIRM); or with TYPE(SYNC_LEVEL), and either the synchronization level is CONFIRM, or it is SYNCPT and the remote program subsequently issued CONFIRM. The local program may respond by issuing CONFIRMED, or by issuing another verb such as SEND_ERROR.
- CONFIRM_DEALLOCATE indicates the remote program has issued DEALLOCATE with TYPE(CONFIRM); or with TYPE(SYNC_LEVEL), and either the synchronization level is CONFIRM, or it is SYNCPT and the remote program subsequently issued CONFIRM. The local program may respond by issuing CONFIRMED, or by issuing another verb such as SEND_ERROR.
- TAKE_SYNCPT indicates the remote program has issued either SYNCPT or PREPARE_FOR_SYNCPT, requesting the local program to respond by issuing SYNCPT in order to perform the sync-point function on all protected resources throughout the transaction. Issuing the SYNCPT verb also causes an affirmative reply to be returned to the remote program if the sync-point function is successful. The program may respond, instead, by issuing a verb other than SYNCPT, such as BACKOUT or SEND_ERROR.
- TAKE_SYNCPT_SEND indicates the remote program has issued PREPARE_TO_RECEIVE with TYPE(SYNC_LEVEL), the synchronization level is SYNCPT, and the remote program subsequently issued SYNCPT or PREPARE_FOR_SYNCPT. The local program may respond by issuing SYNCPT, or by issuing another verb such as BACKOUT or SEND_ERROR.
- TAKE_SYNCPT_DEALLOCATE indicates the remote program has issued DEALLOCATE with TYPE(SYNC_LEVEL), the synchronization level is SYNCPT, and the remote program subsequently issued SYNCPT or PREPARE_FOR_SYNCPT. The local program may respond by issuing SYNCPT, or by issuing another verb such as BACKOUT or SEND_ERROR.

State Changes:

Half-duplex conversations only:

Send state is entered when WHAT_RECEIVED indicates SEND.

RECEIVE_IMMEDIATE

Confirm state is entered when WHAT_RECEIVED indicates CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE.

Sync point state is entered when WHAT_RECEIVED indicates TAKE_SYNCPT, TAKE_SYNCPT_SEND, or TAKE_SYNCPT_DEALLOCATE.

No state change occurs when WHAT_RECEIVED indicates DATA, DATA_COMPLETE, DATA_INCOMPLETE, or LL_TRUNCATED.

Full-duplex conversations only

Reset state is entered when a DEALLOCATE_NORMAL indication is received while in receive-only state, when a DEALLOCATE_ABEND_* return code is received, or when an ALLOCATION_ERROR return code is received.

Send-only state is entered when a DEALLOCATE_NORMAL indication is received while in send-receive state.

Notes:

1. When FILL(LL) is specified, the program is to receive a logical record and there are the following possibilities:

- The program receives a complete logical record or the last remaining portion of a complete record. The length of the record or portion of the record is equal to or less than the length specified on the LENGTH parameter. The WHAT_RECEIVED parameter indicates DATA_COMPLETE.
- The program receives an incomplete logical record. The logical record is incomplete because:
 - The length of the logical record is greater than the length specified on the LENGTH parameter; in this case the amount received equals the length specified.
 - Only a portion of the logical record is available (possibly because it has been truncated), the portion being equal to or less than the length specified on the LENGTH parameter.

The WHAT_RECEIVED parameter indicates DATA_INCOMPLETE. The program issues another RECEIVE_IMMEDIATE (or possibly multiple RECEIVE_IMMEDIATES) to receive the remainder of the logical record.

- The program receives no part of the logical record because it was truncated after the first byte of the LL field. The WHAT_RECEIVED parameter indicates LL_TRUNCATED.

Refer to the SEND_DATA verb for a definition of complete and incomplete logical records.

2. When FILL(BUFFER) is specified, the program is to receive data independent of its logical-record format. The program receives whatever data is available, up to the amount specified on the LENGTH parameter. The program is responsible for tracking the logical-record format of the data.

RECEIVE_IMMEDIATE

3. RECEIVE_IMMEDIATE with LENGTH(0) has no special significance. The type of information available, if any, is indicated by the RETURN_CODE and WHAT_RECEIVED parameters, as usual. If data is available and FILL(LL) is specified, the WHAT_RECEIVED parameter indicates DATA_INCOMPLETE. If data is available and FILL(BUFFER) is specified, the WHAT_RECEIVED parameter indicates DATA. In either case, however, the program receives no data.
4. The program receives only one kind of information at a time. For example, it may receive data or a CONFIRM request, but it does not receive both at the same time. Also, if the remote program truncates a logical record, the local program receives the indication of the truncation on the RECEIVE_IMMEDIATE it issues after receiving all of the truncated record. The RETURN_CODE and WHAT_RECEIVED parameters indicate to the program the kind of information the program receives, if any.
5. No state change has occurred because EXPEDITED_DATA_RECEIVED indicates YES.

When expedited data is received by the LU, it is indicated on the pertinent verbs if the return parameter EXPEDITED_DATA_RECEIVED exists on the verbs. It will continue to be indicated until such time that the data is received by the TP issuing the RECEIVE_EXPEDITED_DATA verb.
6. References in this verb description to a program being in a particular state are only in terms of the specified conversation.

Half-duplex conversations only:

1. RECEIVE_IMMEDIATE resets or cancels posting. If posting is active and the conversation has been posted, posting is reset. If posting is active and the conversation has not been posted, posting is canceled (posting will not occur). See the POST_ON_RECEIPT verb for more details about posting.
2. The REQUEST_TO_SEND notification is usually received when the local transaction program is in send state, and reported to the program on a SEND_DATA verb or on a SEND_ERROR verb issued in send state. However, the notification can be received when the program is in receive state under the following conditions:
 - When the local program just entered receive state and the remote program issued REQUEST_TO_SEND before it entered send state.
 - When the remote program has just entered receive state by means of the PREPARE_TO_RECEIVE verb (not RECEIVE_AND_WAIT), and then issued REQUEST_TO_SEND before the local program enters send state. This can occur because the REQUEST_TO_SEND is transmitted as an expedited request and can therefore arrive ahead of the request carrying the SEND indication. Potentially, the local program cannot distinguish this case from the first. This ambiguity is avoided when the remote program waits until it receives information from the local program before it issues the REQUEST_TO_SEND.
 - When the remote program issues the REQUEST_TO_SEND in send state (see "Notes on Implementation Details" on page A-53).

RECEIVE_IMMEDIATE

3. The REQUEST_TO_SEND notification is returned to the program in addition to (not in place of) the information indicated by the RETURN_CODE and WHAT_RECEIVED parameters.

Full-duplex conversations only:

1. The REQUEST_TO_SEND notification is always set to NO.

REQUEST_TO_SEND

REQUEST_TO_SEND

Notifies the remote program that the local program is requesting to enter send state for the conversation. The conversation will be changed to send state when the local program subsequently receives a SEND indication from the remote program.

REQUEST_TO_SEND	Option No. [Base]	Half-duplex [√]	Full-duplex []
Supplied Parameters:			
RESOURCE (<i>variable</i>)			
WAIT_OBJECT (BLOCKING) (<i>VALUE</i> (<i>variable</i>))			
Returned Parameters:			
RETURN_CODE (<i>variable</i>)			

Supplied Parameters:

RESOURCE specifies the variable containing the resource ID of the conversation.

WAIT_OBJECT specifies whether the verb should be processed in a blocking or nonblocking manner. The default value is a blocking verb operation. If a non-blocking verb operation is to be requested, then

WAIT_OBJECT(VALUE(variable)) must be specified.

- **BLOCKING** specifies this is a blocking verb operation.
- **VALUE(variable)** specifies a nonblocking verb operation of one of two types:
 - a *null* variable specifies to complete the verb operation but without notification.
 - a nonnull *variable* specifies to complete the verb operation but with notification. The variable represents a wait object. Its use here associates it with completion of this verb. It can be listed as an entry in the **WAIT_OBJECT_LIST** in a subsequent **WAIT_FOR_COMPLETION** verb for notification (posting) purposes. Origination of this variable (by the TP or the operating system) is implementation dependent.

Returned Parameters:

REQUEST_TO_SEND

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- OK
- CONVERSATION_ENDED
- OPERATION_INCOMPLETE
- OPERATION_NOT_ACCEPTED
- PROGRAM_PARAMETER_CHECK, for one of the following reasons:
 - RESOURCE specifies an unassigned resource ID.
 - This verb is not supported for full-duplex conversations.
- PROGRAM_STATE_CHECK, for one of the following reasons:
 - The conversation is not in receive, confirm, send, or sync-point state.
 - The conversation is in deallocate-pending state.

State Changes:

None

Notes:

1. The REQUEST_TO_SEND notification is indicated to the remote program by means of the REQUEST_TO_SEND_RECEIVED parameter. When the REQUEST_TO_SEND_RECEIVED parameter is set to YES, the remote program is requested to enter receive state and thereby place the local program in send state. A program enters receive state by means of the RECEIVE_AND_WAIT or PREPARE_TO_RECEIVE verb. The partner program enters the corresponding send state when it issues a RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb and receives the SEND indication (on the WHAT_RECEIVED parameter).
2. The REQUEST_TO_SEND_RECEIVED indication of YES is normally returned to the remote program when it is in send state, that is, on a SEND_DATA verb or on a SEND_ERROR verb issued in send state. However, it can be returned on a RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb; see the description of RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE for details about when this can occur.
3. When the remote LU receives the REQUEST_TO_SEND notification, it retains the notification until the remote program issues a verb on which the notification can be indicated, that is, a verb with the REQUEST_TO_SEND_RECEIVED parameter. The remote LU will retain only one REQUEST_TO_SEND notification at a time (per conversation); additional notifications are discarded until the retained notification is indicated to the remote program. It is therefore possible for the local program to issue the REQUEST_TO_SEND verb more times than are indicated to the remote program.
4. The REQUEST_TO_SEND verb, as described in this book, may be issued only when the conversation is in send, receive, confirm, or sync-point state. Issuing the verbs for a conversation in any other state is described as a state-check ABEND condition. As an alternative to the ABEND condition, a product may

REQUEST_TO_SEND

also permit its transaction programs to issue the verbs for conversations in send or defer state.

5. References in this verb description to a program being in a particular state are only in terms of the specified conversation.
6. If a normal conversation deallocation occurs when a REQUEST_TO_SEND verb is outstanding, a return code of CONVERSATION_ENDED is returned. CONVERSATION_ENDED is also returned when a DEALLOCATE(ABEND), allocation error, or conversation failure occurs. This return code indicates that a REQUEST_TO_SEND verb was outstanding when the conversation ended, the request was not completed successfully, and the expedited-send queue is effectively closed.

No state change occurs because CONVERSATION_ENDED is returned. Subsequent REQUEST_TO_SEND verbs will not be performed, but will be rejected with the CONVERSATION_ENDED return code.

SEND_DATA

Sends data to the remote transaction program. The data format consists of logical records. The amount of data is specified independently of the data format.

SEND_DATA	<i>Option No.</i> [Base]	<i>Half-duplex</i> [✓]	<i>Full-duplex</i> [✓]
Supplied Parameters:			
RESOURCE (<i>variable</i>) DATA (<i>variable</i>) LENGTH (<i>variable</i>)			
ENCRYPT (<i>NO</i>) (<i>YES</i>)			
WAIT_OBJECT (<i>BLOCKING</i>) (<i>VALUE</i> (<i>variable</i>))			
Returned Parameters:			
RETURN_CODE (<i>variable</i>) REQUEST_TO_SEND_RECEIVED (<i>variable</i>) EXPEDITED_DATA_RECEIVED (<i>variable</i>)			

Supplied Parameters:

RESOURCE specifies the variable containing the resource ID of the conversation on which the data is to be sent.

DATA specifies the variable containing the data to be sent. The data consists of logical records. Each logical record consists of a two-byte length field (denoted as LL) followed by a data field; the length of the data field can range from zero to 32765 bytes. The two-byte length field contains the 15-bit binary length of the record, and a high-order bit that is not examined by the LU (it is used, for example, by the LU's mapped conversation component in support of the mapped conversation verbs). The length of the record includes the two-byte length field, that is, it equals the length of the data field plus two. Thus, logical-record length values of hex¹ 0000, 0001, 8000, and 8001, are invalid.

¹ Hex (meaning hexadecimal) refers to the base-16 numbering system.

SEND_DATA

LENGTH specifies the variable containing the length of the data to be sent. This data length is not related in any way to the length of a logical record. It is used only to determine the length of the data located at the variable specified by the DATA parameter.

The data length may be zero or greater. If zero, no data is sent for this issuance of the verb, and the DATA parameter is not significant. However, the other parameters are significant and retain their meaning as described.

ENCRYPT specifies encryption is to be used when the mode allows selective encryption.

- **NO** specifies encryption is not required.
- **YES** specifies encryption is required.

If the encryption for the mode name is none, **ENCRYPTION_NOT_SUPPORTED** will be returned. If the encryption for the mode name is mandatory, the **ENCRYPT** parameter is ignored, as all data is encrypted. If the mode allows selective encryption, the action by the LU depends on the result of the negotiation with the remote LU at session activation:

- If selective encryption was agreed upon, the default value of the parameter is **NO**.
- If mandatory encryption was agreed upon, the parameter is ignored, as all data is encrypted.

WAIT_OBJECT specifies whether the verb should be processed in a blocking or nonblocking manner. The default value is a blocking verb operation. If a non-blocking verb operation is to be requested, then **WAIT_OBJECT(VALUE(variable))** must be specified.

- **BLOCKING** specifies this is a blocking verb operation.
- **VALUE(variable)** specifies a nonblocking verb operation of one of two types:
 - a *null* variable specifies to complete the verb operation but without notification.
 - a nonnull *variable* specifies to complete the verb operation but with notification. The variable represents a wait object. Its use here associates it with completion of this verb. It can be listed as an entry in the **WAIT_OBJECT_LIST** in a subsequent **WAIT_FOR_COMPLETION** verb for notification (posting) purposes. Origination of this variable (by the TP or the operating system) is implementation dependent.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- **OK**
- **ENCRYPTION_NOT_SUPPORTED**
- **OPERATION_INCOMPLETE**
- **OPERATION_NOT_ACCEPTED**
- **PROGRAM_PARAMETER_CHECK**, for one of the following reasons:

- RESOURCE specifies an unassigned resource ID.
- DATA contains an invalid logical record length (LL) value of hex 0000, 0001, 8000, or 8001.

Return codes for half-duplex conversations only:

- ALLOCATION_ERROR
- BACKED_OUT
- DEALLOCATE_ABEND_PROG
- DEALLOCATE_ABEND_SVC
- DEALLOCATE_ABEND_TIMER
- PROG_ERROR_PURGING
- PROGRAM_STATE_CHECK, for the following reason:
 - The conversation is not in send state.
- RESOURCE_FAILURE_NO_RETRY
- RESOURCE_FAILURE_RETRY
- SVC_ERROR_PURGING

Return codes for full-duplex conversations only:

- ERROR_INDICATION, with one of the following subcodes: [Refer to Appendix F, "Conversation Return Codes" on page F-1 for a complete list of subcodes with their explanations.]
 - ALLOCATION_ERROR_PENDING
 - DEALLOCATE_ABEND_PROG_PENDING
 - DEALLOCATE_ABEND_SVC_PENDING
 - DEALLOCATE_ABEND_TIMER_PENDING
 - RESOURCE_FAILURE_NO_RETRY_PENDING
 - RESOURCE_FAILURE_RETRY_PENDING
 - UNKNOWN_ERROR_TYPE_PENDING
- PROGRAM_STATE_CHECK, for the following reason:
 - The conversation is not in send-receive or send-only state.

REQUEST_TO_SEND_RECEIVED specifies the variable in which is returned an indication of whether REQUEST_TO_SEND has been received.

- YES indicates a REQUEST_TO_SEND notification has been received from the remote transaction program. The remote program has issued REQUEST_TO_SEND, requesting the local program to enter receive state and thereby place the remote program in send state.
- NO indicates a REQUEST_TO_SEND notification has not been received.

This parameter is always set to NO for full-duplex conversations.

EXPEDITED_DATA_RECEIVED specifies the variable in which is returned an indication of whether SEND_EXPEDITED_DATA has been received.

- YES indicates a SEND_EXPEDITED_DATA notification has been received from the remote transaction program.
- NO indicates a SEND_EXPEDITED_DATA notification has not been received from the remote transaction program.

State Changes:

SEND_DATA

Half-duplex conversations only:

None

Full-duplex conversations only:

Reset state is entered when an ERROR_INDICATION return code is returned while in send-only state.

Notes:

1. The data sent by the program consists of logical records. The logical records are independent of the length of data as specified by the LENGTH parameter. That is, the data may consist of one or more complete records, the beginning of a record, the middle of a record, or the end of a record. The following combinations of these are also possible:
 - One or more complete records, followed by the beginning of a record.
 - The end of a record, followed by one or more complete records.
 - The end of a record, followed by one or more complete records, followed by the beginning of a record.
 - The end of a record, followed by the beginning of a record.
2. The program must finish sending a logical record before issuing any of the following verbs:
 - CONFIRM
 - DEALLOCATE with TYPE(FLUSH), TYPE(CONFIRM), or TYPE(SYNC_LEVEL)
 - PREPARE_TO_RECEIVE
 - RECEIVE_AND_WAIT
 - SYNCPT

A program finishes sending a logical record when it sends a complete record or when it truncates an incomplete record.
3. A complete logical record contains the two-byte LL field and all bytes of the data field, as determined by the logical-record length. (If the data field is of zero length, the complete logical record contains only the two-byte length field.) An incomplete logical record consists of any amount of data less than a complete record. It can consist of only the first byte of the LL field, the two-byte LL field plus all of the data field except the last byte, or any amount in between. A logical record is incomplete until the last byte of the data field is sent, or until the second byte of the LL field is sent if the data field is of zero length.
4. A program can truncate an incomplete logical record by issuing the SEND_ERROR verb. SEND_ERROR causes the LU to flush its send buffer, which includes sending the truncated record. The LU then treats the first two bytes of data specified in the next SEND_DATA as the LL field. Issuing DEALLOCATE with TYPE(ABEND_PROG), TYPE(ABEND_SVC), or TYPE(ABEND_TIMER) also truncates an incomplete logical record.

5. The LU buffers the data to be sent to the remote LU until it accumulates from one or more SEND_DATAs a sufficient amount for transmission, or until the local program issues a verb that causes the LU to flush its send buffer. The amount of data that is sufficient for transmission depends on the characteristics of the session allocated for the conversation, and can vary from one session to another.
6. No state change has occurred because EXPEDITED_DATA_RECEIVED indicates YES.

When expedited data is received by the LU, it is indicated on the pertinent verbs if the return parameter EXPEDITED_DATA_RECEIVED exists on the verbs. It will continue to be indicated until such time that the data is received by the TP issuing the RECEIVE_EXPEDITED_DATA verb.
7. References in this verb description to a program being in a particular state are only in terms of the specified conversation.

Half-duplex conversations only:

1. When REQUEST_TO_SEND_RECEIVED indicates YES, the remote program is requesting the local program to enter receive state and thereby place the remote program in send state. A program enters receive state by means of the RECEIVE_AND_WAIT or PREPARE_TO_RECEIVE verb. The partner program enters the corresponding send state when it issues a RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb and receives the SEND indication (on the WHAT_RECEIVED parameter).

Full-duplex conversations only:

1. The REQUEST_TO_SEND notification is always set to NO.
2. An ERROR_INDICATION return code is returned when a terminating error has been received from the remote program; the SEND_DATA request is not performed. The terminating error may be a DEALLOCATE_ABEND, an ALLOCATION_ERROR, or a conversation failure. The nature of the terminating error is reflected in a subcode to the ERROR_INDICATION return code. If the nature of the terminating error is not known, a subcode of UNKNOWN_ERROR_TYPE_PENDING is returned.

When an ERROR_INDICATION return code is returned while the conversation state is send-only, the conversation state will change to reset. If an ERROR_INDICATION is returned while the conversation state is send-receive, no state change occurs; however, the send queue is effectively closed and subsequent SEND_DATA verbs will be rejected with the ERROR_INDICATION return code.

When an ERROR_INDICATION is returned while the conversation state is send-receive, the local program may end the conversation by either issuing receive queue verbs until the return code for the terminating error is received, or by issuing a DEALLOCATE with TYPE(ABEND).

3. When the subcode associated with the ERROR_INDICATION return code is ALLOCATION_ERROR, no additional information is returned regarding the nature of the ALLOCATION_ERROR.

SEND_ERROR

SEND_ERROR

Informs the remote transaction program that the local program detected an error. If the conversation is in send state, the LU flushes its send buffer.

For half-duplex conversations:

Upon successful completion of this verb, the local program is in send state and the remote program is in receive state. Further action is defined by transaction program logic.

For full-duplex conversations:

No state changes occur. SEND_ERROR applies to the send direction only. The error notification will be reported to the partner's receive queue.

SEND_ERROR	Option No. [Base]	Half-duplex [√]	Full-duplex [√]
Supplied Parameters:			
RESOURCE (<i>variable</i>)			
TYPE (<i>PROG</i>) (<i>SVC</i>)		■ 291	
LOG_DATA (<i>NO</i>) (<i>YES</i> (<i>variable</i>))		■ 290	
WAIT_OBJECT (<i>BLOCKING</i>) (<i>VALUE</i> (<i>variable</i>))		■ 113	
Returned Parameters:			
RETURN_CODE (<i>variable</i>)			
REQUEST_TO_SEND_RECEIVED (<i>variable</i>)			
EXPEDITED_DATA_RECEIVED (<i>variable</i>)		■ 112	

Supplied Parameters:

RESOURCE specifies the variable containing the resource ID of the conversation.

TYPE specifies the level of error—application or service—being reported. This parameter is intended to distinguish between errors to be reported to end-user application transaction programs and errors to be reported to LU services transaction programs.

- **PROG** specifies an end-user application program error is being reported. For instance, this error type is used by the LU services component for mapped conversation verbs in its processing of the MC_SEND_ERROR verb. The corresponding component at the remote LU will pass the error return code on to the remote end-user application program.
- **SVC** specifies an LU services error is being reported. For instance, this error type is used by the LU services component for mapped conversation verbs to report errors detected within the LU services layer.

LOG_DATA specifies whether product-unique error information is to be placed in the system error logs of the LUs supporting this conversation.

- **NO** specifies that no error information is to be placed in the system error logs.
- **YES** specifies that product-unique error information is to be placed in the system error logs of the local and remote LUs. The specified variable contains the product-unique error information, in the format of the Error Log GDS variable. See *SNA LU 6.2 Reference: Peer Protocols* and *SNA Formats* for a definition of the Error Log GDS variable.

WAIT_OBJECT specifies whether the verb should be processed in a blocking or nonblocking manner. The default value is a blocking verb operation. If a non-blocking verb operation is to be requested, then **WAIT_OBJECT(VALUE(variable))** must be specified.

- **BLOCKING** specifies this is a blocking verb operation.
- **VALUE(variable)** specifies a nonblocking verb operation of one of two types:
 - a *null* variable specifies to complete the verb operation but without notification.
 - a nonnull *variable* specifies to complete the verb operation but with notification. The variable represents a wait object. Its use here associates it with completion of this verb. It can be listed as an entry in the **WAIT_OBJECT_LIST** in a subsequent **WAIT_FOR_COMPLETION** verb for notification (posting) purposes. Origination of this variable (by the TP or the operating system) is implementation dependent.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program. The return codes that can be returned depend on the state of the conversation at the time this verb is issued:

- **OK**
- **OPERATION_INCOMPLETE**
- **OPERATION_NOT_ACCEPTED**
- **PROGRAM_PARAMETER_CHECK**
 - **RESOURCE** specifies an unassigned resource ID.

Return codes for half-duplex conversations only:

- If the **SEND_ERROR** is issued in send state, the following return codes can be returned:

SEND_ERROR

- ALLOCATION_ERROR
 - BACKED_OUT
 - DEALLOCATE_ABEND_PROG
 - DEALLOCATE_ABEND_SVC
 - DEALLOCATE_ABEND_TIMER
 - PROG_ERROR_PURGING
 - RESOURCE_FAILURE_NO_RETRY
 - RESOURCE_FAILURE_RETRY
 - SVC_ERROR_PURGING
- If the SEND_ERROR is issued in receive state, the following return codes can be returned:
 - OK
 - DEALLOCATE_NORMAL
 - RESOURCE_FAILURE_NO_RETRY
 - RESOURCE_FAILURE_RETRY
 - If the SEND_ERROR is issued in confirm state or sync-point state, the following return codes can be returned:
 - OK
 - RESOURCE_FAILURE_NO_RETRY
 - RESOURCE_FAILURE_RETRY
 - If the SEND_ERROR is issued in any state other than send, receive, confirm, or sync-point state, the following return codes can be returned:
 - PROGRAM_STATE_CHECK
 - The conversation is not in send, receive, confirm, or sync-point state.

Return codes for full-duplex conversations only:

- ERROR_INDICATION, with one of the following subcodes: [Refer to Appendix F, "Conversation Return Codes" on page F-1 for a complete list of subcodes with their explanations.]
 - ALLOCATION_ERROR_PENDING
 - DEALLOCATE_ABEND_PROG_PENDING
 - DEALLOCATE_ABEND_SVC_PENDING
 - DEALLOCATE_ABEND_TIMER_PENDING
 - RESOURCE_FAILURE_NO_RETRY_PENDING
 - RESOURCE_FAILURE_RETRY_PENDING
 - UNKNOWN_ERROR_TYPE_PENDING
- PROGRAM_STATE_CHECK
 - The conversation is not in send-receive or send-only state.

REQUEST_TO_SEND_RECEIVED specifies the variable in which is returned an indication of whether REQUEST_TO_SEND has been received.

- YES indicates a REQUEST_TO_SEND notification has been received from the remote transaction program. The remote program has issued REQUEST_TO_SEND, requesting the local program to enter receive state and thereby place the remote program in send state.
- NO indicates a REQUEST_TO_SEND notification has not been received.

This parameter is always set to NO for full-duplex conversations.

EXPEDITED_DATA_RECEIVED specifies the variable in which is returned an indication of whether **SEND_EXPEDITED_DATA** has been received.

- YES indicates a **SEND_EXPEDITED_DATA** notification has been received from the remote transaction program.
- NO indicates a **SEND_EXPEDITED_DATA** notification has not been received from the remote transaction program.

State Changes:

Half-duplex conversations only:

Send state is entered when the verb is issued in receive, confirm, or sync-point state.

No state change occurs when the verb is issued in send state.

Full-duplex conversations only:

Reset state is entered when an **ERROR_INDICATION** return code is returned while in send-only state.

Notes:

1. The LU may send the error notification to the remote LU immediately, that is, during the processing of this verb, or the LU may defer sending the notification until a later time. The determination is made as follows:
 - If the local product does not support the **FLUSH** verb (see “Notes on Implementation Details” on page A-53), then the LU sends the error notification immediately.
 - If the local product does support the **FLUSH** verb, then the LU may or may not send the notification immediately, depending on the product. If the LU defers sending the notification, it buffers the notification until it accumulates a sufficient amount of information for transmission, or until the local program issues a verb that causes the LU to flush its send buffer. The amount of information that is sufficient for transmission depends on the characteristics of the session allocated for the conversation, and can vary from one session to another. Transmission of the information may begin immediately, if the **LOG_DATA** parameter is specified with sufficient log data, or transmission may not begin until sufficient data from subsequent **SEND_DATA** verbs is also buffered.
2. The local program can ensure that the remote program receives the error notification as soon as possible by issuing **FLUSH** immediately after **SEND_ERROR**.
3. The program may use this verb for various application-level functions. For example, the program may issue this verb to truncate an incomplete logical record it is sending, to inform the remote program of an error it detected in data it received, or to reject a confirmation or sync-point request.

SEND_ERROR

4. No state change has occurred because EXPEDITED_DATA_RECEIVED indicates YES.

When expedited data is received by the LU, it is indicated on the pertinent verbs if the return parameter EXPEDITED_DATA_RECEIVED exists on the verbs. It will continue to be indicated until such time that the data is received by the TP issuing the RECEIVE_EXPEDITED_DATA verb.

5. References in this verb description to a program being in a particular state are only in terms of the specified conversation.

Half-duplex conversations only:

1. SEND_ERROR is reported to the remote transaction program as one of the following return codes (based on the TYPE parameter):
 - PROG_ERROR_TRUNC or SVC_ERROR_TRUNC - The local program issued SEND_ERROR in send state after sending an incomplete logical record (see SEND_DATA). The record has been truncated.
 - PROG_ERROR_NO_TRUNC or SVC_ERROR_NO_TRUNC - The local program issued SEND_ERROR in send state after sending a complete logical record (see SEND_DATA) or prior to sending any record. No truncation has occurred.
 - PROG_ERROR_PURGING or SVC_ERROR_PURGING - The local program issued SEND_ERROR in receive state and all information sent by the remote program and not yet received by the local program, if any, has been purged; or the local program issued SEND_ERROR in confirm or sync-point state, in which case no purging has occurred.
2. When SEND_ERROR is issued in receive state, purging of incoming information occurs. The incoming information that is purged includes the following return code indications:
 - ALLOCATION_ERROR
 - BACKED_OUT
 - DEALLOCATE_ABEND_PROG
 - DEALLOCATE_ABEND_SVC
 - DEALLOCATE_ABEND_TIMER
 - PROG_ERROR_NO_TRUNC
 - PROG_ERROR_PURGING
 - PROG_ERROR_TRUNC
 - SVC_ERROR_NO_TRUNC
 - SVC_ERROR_PURGING
 - SVC_ERROR_TRUNC

The return code DEALLOCATE_NORMAL is reported instead of ALLOCATION_ERROR, DEALLOCATE_ABEND_PROG, DEALLOCATE_ABEND_SVC, or DEALLOCATE_ABEND_TIMER. If a DEALLOCATE_NORMAL return code to SEND_ERROR is received on a conversation allocated with SYNC_LEVEL(SYNCPT), the LU assumes that a DEALLOCATE_ABEND_* has occurred, even though the return code indication was purged, since DEALLOCATE TYPE(FLUSH) or TYPE(CONFIRM) are not allowed on protected conversations. As a result,

other protected conversations of that program will be put in the backout-required state.

The return code OK is reported instead of the other return codes. When the return code BACKED_OUT is purged, the remote LU resends the BACKED_OUT indication and the local program receives the return code on a subsequent verb.

The other kinds of incoming information that are purged are:

- Data, sent by means of the SEND_DATA verb.
- Confirmation request, sent by means of the CONFIRM, PREPARE_TO_RECEIVE, or DEALLOCATE verb.
- Sync-point request, sent by means of the SYNCPT, PREPARE_TO_RECEIVE, or DEALLOCATE verb.

If the confirmation or sync-point request was sent in conjunction with the DEALLOCATE verb (by means of its TYPE(CONFIRM) or TYPE(SYNC_LEVEL) parameter), the deallocation request is also purged.

Incoming information that is not purged is the REQUEST_TO_SEND indication. This indication is reported to the program when it issues a verb that includes the REQUEST_TO_SEND_RECEIVED parameter.

3. When REQUEST_TO_SEND_RECEIVED indicates YES, the remote program is requesting the local program to enter receive state and thereby place the remote program in send state. A program enters receive state by means of the RECEIVE_AND_WAIT or PREPARE_TO_RECEIVE verb. The partner program enters the corresponding send state when it issues a RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb and receives the SEND indication (on the WHAT_RECEIVED parameter).
4. SEND_ERROR resets or cancels posting. If posting is active and the conversation has been posted, posting is reset. If posting is active and the conversation has not been posted, posting is canceled (posting will not occur). See the POST_ON_RECEIPT verb for more details about posting.

Full-duplex conversations only:

1. The REQUEST_TO_SEND notification is always set to NO.
2. For full-duplex conversations, SEND_ERROR is used to report an error that occurred on the local program's send queue. The error is reported to the remote transaction program's receive queue as one of the following return codes (based on the TYPE parameter):
 - PROG_ERROR_TRUNC or SVC_ERROR_TRUNC - The local program issued SEND_ERROR in send-receive or send-only state after sending an incomplete logical record (see SEND_DATA). The record has been truncated.
 - PROG_ERROR_NO_TRUNC or SVC_ERROR_NO_TRUNC - The local program issued SEND_ERROR in send-receive or send-only state after sending a complete logical record (see SEND_DATA) or prior to sending any record. No truncation has occurred.

SEND_ERROR

3. SEND_ERROR is not supported in the receive direction; therefore, there is no equivalent to SEND_ERROR issued from receive state for half-duplex conversations. Purging of incoming information does not take place.
4. An ERROR_INDICATION return code is returned when a terminating error has been received from the remote program, and the SEND_ERROR verb is not performed. The terminating error may be a DEALLOCATE_ABEND, an ALLOCATION_ERROR, or a conversation failure. The nature of the terminating error is reflected in a subcode to the ERROR_INDICATION return code. If the nature of the terminating error is not known, a subcode of UNKNOWN_ERROR_TYPE_PENDING is returned.

When an ERROR_INDICATION return code is returned while the conversation state is send-only, the conversation state will change to reset. If an ERROR_INDICATION is returned while the conversation state is send-receive, no state change occurs; however, the send queue is effectively closed and subsequent SEND_ERROR verbs will be rejected with the ERROR_INDICATION return code.

When an ERROR_INDICATION is returned while the conversation state is send-receive, the local program may end the conversation by either issuing receive queue verbs until the return code for the terminating error is received, or by issuing a DEALLOCATE with TYPE(ABEND).

5. When the subcode associated with the ERROR_INDICATION return code is ALLOCATION_ERROR, no additional information is returned regarding the nature of the ALLOCATION_ERROR.

SEND_EXPEDITED_DATA

Sends data to the remote transaction program in an expedited manner. This means it may arrive at the remote transaction program before data sent earlier via a send queue verb, e.g., SEND_DATA. (See "Nonblocking Support for LU 6.2 Conversation Verbs" on page 3-9 for a complete list of send queue verbs.)

At the remote transaction program, a RECEIVE_EXPEDITED_DATA verb is used to receive this data.

SEND_EXPEDITED_DATA	<i>Option No.</i> [112]	<i>Half-duplex</i> [✓]	<i>Full-duplex</i> [✓]
Supplied Parameters: RESOURCE (<i>variable</i>) DATA (<i>variable</i>) LENGTH (<i>variable</i>) WAIT_OBJECT (<i>BLOCKING</i>) (<i>VALUE</i> (<i>variable</i>))			
Returned Parameters: RETURN_CODE (<i>variable</i>) REQUEST_TO_SEND_RECEIVED (<i>variable</i>) EXPEDITED_DATA_RECEIVED (<i>variable</i>)			

113

Supplied Parameters:

RESOURCE specifies the variable containing the resource ID.

DATA specifies the variable containing the data record to be sent. The data record consists entirely of data and is not in logical record format; no length field (LL) is present.

LENGTH specifies the variable containing the length (in bytes) of the data record to be sent. The valid length is 1-86. If the value specified for the LENGTH field is not in the range 1-86, an error return code PROGRAM_PARAMETER_CHECK is returned to the SEND_EXPEDITED_DATA verb.

The amount of DATA to be sent is determined by the length value specified. If the amount of DATA to be sent is greater than the length specified, the extra data is ignored.

SEND_EXPEDITED_DATA

WAIT_OBJECT specifies whether the verb should be processed in a blocking or nonblocking manner. The default value is a blocking verb operation. If a non-blocking verb operation is to be requested, then **WAIT_OBJECT(VALUE(variable))** must be specified.

- **BLOCKING** specifies this is a blocking verb operation.
- **VALUE(variable)** specifies a nonblocking verb operation of one of two types:
 - a *null* variable specifies to complete the verb operation but without notification.
 - a nonnull *variable* specifies to complete the verb operation but with notification. The variable represents a wait object. Its use here associates it with completion of this verb. It can be listed as an entry in the **WAIT_OBJECT_LIST** in a subsequent **WAIT_FOR_COMPLETION** verb for notification (posting) purposes. Origination of this variable (by the TP or the operating system) is implementation dependent.

Returned Parameters:

RETURN_CODE specifies the variable in which a return code is returned to the local program. The return code indicates the result of verb execution. The verb may be issued in any state except in reset state.

- OK
- CONVERSATION_ENDED
- EXPEDITED_DATA_NOT_SUPPORTED_BY_LU
- OPERATION_INCOMPLETE
- OPERATION_NOT_ACCEPTED
- PROGRAM_PARAMETER_CHECK, for one of the following reasons:
 - RESOURCE specifies an unassigned resource ID
 - LENGTH specifies an invalid length value has been used that is not in the range 1-86.
 - Expedited data is not supported
- PROGRAM_STATE_CHECK, for the following reason:
 - The conversation is in deallocate-pending state.

REQUEST_TO_SEND_RECEIVED specifies the variable in which is returned an indication of whether **REQUEST_TO_SEND** has been received.

- YES indicates a **REQUEST_TO_SEND** notification has been received from the remote transaction program.
- NO indicates a **REQUEST_TO_SEND** notification has not been received from the remote transaction program.

This parameter is always set to NO for full-duplex conversations.

EXPEDITED_DATA_RECEIVED specifies the variable in which is returned an indication of whether **SEND_EXPEDITED_DATA** has been received.

- YES indicates a **SEND_EXPEDITED_DATA** notification has been received from the remote transaction program.
- NO indicates a **SEND_EXPEDITED_DATA** notification has not been received from the remote transaction program.

State Changes:

None

Notes:

1. The remote LU retains the expedited data until the remote program issues a RECEIVE_EXPEDITED_DATA verb.
2. It is the implementation's choice to allow the remote LU to retain one or more than one expedited data at a time (per conversation). However, the not-yet-received expedited data must not be overwritten by the new one. The implementation should not send +RSP(EXPD) to the remote LU unless an expedited-flow buffer is available to receive the next expedited data (if any).
3. If a normal conversation deallocation occurs when a SEND_EXPEDITED_DATA verb is outstanding, a return code of CONVERSATION_ENDED is returned. CONVERSATION_ENDED is also returned when a DEALLOCATE(ABEND), allocation error, or conversation failure occurs. This return code indicates that a SEND_EXPEDITED_DATA verb was outstanding when the conversation ended, the verb was not completed successfully, and the expedited-send queue is effectively closed.

No state change occurs because CONVERSATION_ENDED is returned. Subsequent SEND_EXPEDITED_DATA verbs will not be performed, but will be rejected with the CONVERSATION_ENDED return code.
4. If a SEND_EXPEDITED_DATA verb is issued from the local program, a RECEIVE_EXPEDITED_DATA verb needs to be issued by the remote program to receive the expedited data.

Full-duplex conversations only:

1. The REQUEST_TO_SEND notification is always set to NO.

TEST

TEST

Tests the specified conversation for a condition. The return code indicates the result of the test.

TEST	Option No.	Half-duplex	Full-duplex
	[103,245]	[√]	[]
Supplied Parameters:			
RESOURCE (variable)			
TEST (POSTED)			
(REQUEST_TO_SEND_RECEIVED)			
Returned Parameters:			
RETURN_CODE (variable)			

Supplied Parameters:

RESOURCE specifies the variable containing the resource ID of the conversation.

TEST specifies the condition to be tested.

- **POSTED** specifies to test whether the conversation has been posted. The return code indicates whether posting has occurred.
- **REQUEST_TO_SEND_RECEIVED** specifies to test whether REQUEST_TO_SEND notification has been received from the remote transaction program. The return code indicates whether the notification has been received.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program. The return code indicates the result of the test. The TEST parameter determines which of the following return codes can be returned to the program.

- If TEST(POSTED) is specified, one of the following return codes is returned:
 - OK
 - DATA
 - NOT_DATA
 - POSTING_NOT_ACTIVE
 - UNSUCCESSFUL

- ALLOCATION_ERROR
 - BACKED_OUT
 - DEALLOCATE_ABEND_PROG
 - DEALLOCATE_ABEND_SVC
 - DEALLOCATE_ABEND_TIMER
 - DEALLOCATE_NORMAL
 - PROG_ERROR_NO_TRUNC
 - PROG_ERROR_PURGING
 - PROG_ERROR_TRUNC
 - SVC_ERROR_NO_TRUNC
 - SVC_ERROR_PURGING
 - SVC_ERROR_TRUNC
 - RESOURCE_FAILURE_NO_RETRY
 - RESOURCE_FAILURE_RETRY
 - PROGRAM_PARAMETER_CHECK, for one of the following reasons:
 - RESOURCE specifies an unassigned resource ID.
 - This verb is not supported for full-duplex conversations.
 - PROGRAM_STATE_CHECK, for the following reason:
 - TEST(POSTED) is specified and the conversation is not in receive state.
- If TEST(REQUEST_TO_SEND_RECEIVED) is specified, one of the following return codes is returned:
 - OK
 - UNSUCCESSFUL
 - PROGRAM_PARAMETER_CHECK, for one of the following reasons:
 - RESOURCE specifies an unassigned resource ID.
 - This verb is not supported for full-duplex conversations.
 - PROGRAM_STATE_CHECK, for the following reason:
 - TEST(REQUEST_TO_SEND_RECEIVED) is specified and the conversation is not in send, defer, or receive state.

State Changes:

None

Notes:

1. The TEST(POSTED) parameter on this verb is intended to be used in conjunction with POST_ON_RECEIPT. The use of POST_ON_RECEIPT and this verb allows a program to continue its processing while waiting for information to become available, where the program issues POST_ON_RECEIPT for one or more conversations and then issues this verb for each conversation to determine when information is available to be received.
2. For TEST(POSTED), the return code indicates whether posting has occurred, as follows:
 - OK indicates posting was active for the conversation and it has been posted. Posting is now reset. The subcode of the OK return code indicates why the conversation has been posted.

TEST

- DATA indicates data is available for the program to receive.
- NOT_DATA indicates information other than data, such as a SEND, CONFIRM, or TAKE_SYNCPT indication, is available for the program to receive.

The program should issue RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE in order to receive the information. The program may use the subcode to determine whether it needs to specify the DATA parameter on the RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb.

- POSTING_NOT_ACTIVE indicates posting is not active for the conversation.
- UNSUCCESSFUL indicates posting is active for the conversation and it has not been posted. Posting remains active.

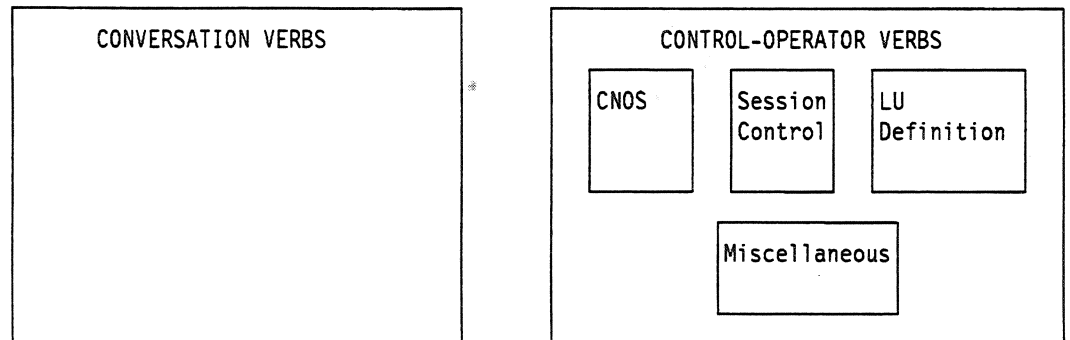
The remaining return codes indicate posting was active for the conversation and it has been posted for the reason indicated by the specific return code. Posting is now reset.

3. Posting is active for a conversation when POST_ON_RECEIPT has been issued for the conversation and posting has not been reset or canceled (see the POST_ON_RECEIPT verb).
4. The TEST(REQUEST_TO_SEND_RECEIVED) parameter specifies to test whether REQUEST_TO_SEND notification has been received from the remote transaction program. The return code indicates whether the notification has been received, as follows:
 - OK indicates REQUEST_TO_SEND has been received. The remote program has issued REQUEST_TO_SEND, requesting the local program to enter receive state and thereby place the remote program in send state. A program enters receive state by means of the RECEIVE_AND_WAIT or PREPARE_TO_RECEIVE verb. The partner program enters the corresponding send state when it issues a RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb and receives the SEND indication (on the WHAT_RECEIVED parameter).
 - UNSUCCESSFUL indicates REQUEST_TO_SEND has not been received.
5. References in this verb description to a program being in a particular state are only in terms of the specified conversation.

Part 3. Control Operator Verbs

Chapter 8. Change Number of Session (CNOS) Verbs	8-1
CNOS Transactions	8-1
Single-Session Connections	8-1
Parallel-Session Connections	8-1
CNOS Service Transaction Program	8-2
CHANGE_SESSION_LIMIT	8-3
INITIALIZE_SESSION_LIMIT	8-8
PROCESS_SESSION_LIMIT	8-13
RESET_SESSION_LIMIT	8-15
 Chapter 9. Session Control Verbs	9-1
Session Control Functions	9-1
ACTIVATE_SESSION	9-2
DEACTIVATE_CONVERSATION_GROUP	9-4
DEACTIVATE_SESSION	9-6
 Chapter 10. LU Definition Verbs	10-1
LU Definition Functions	10-1
DEFINE_LOCAL_LU	10-3
DEFINE_MODE	10-9
DEFINE_REMOTE_LU	10-15
DEFINE_TP	10-20
DELETE	10-28
DISPLAY_LOCAL_LU	10-30
DISPLAY_MODE	10-33
DISPLAY_REMOTE_LU	10-37
DISPLAY_TP	10-39
 Chapter 11. Miscellaneous COPR Verbs	11-1
Miscellaneous Verb Functions	11-1
Persistent Verification	11-1
DISPLAY_SIGNED_ON_LIST	11-3
PROCESS_SIGN_OFF	11-5
SIGNOFF	11-7

Control Operator Verbs



This part describes the category of verbs called *control operator verbs*.² The control operator verbs define the protocol boundary for control operator functions. In particular, the control operator protocol boundary is intended for use by programs that assist the network control operator in performing functions related to the control of an LU and logical interconnections (sessions) among LUs.

See Appendix G, “Control-Operator Return Codes” on page G-1 for a description of the return codes that apply to the control operator verbs.

The control operator verbs are divided into the following subcategories:

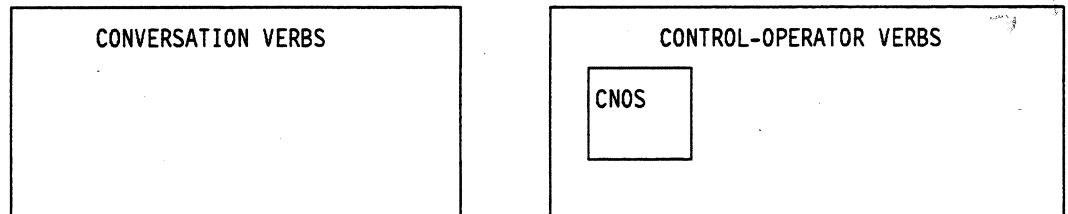
- Change number of sessions verbs
- Session control verbs
- LU definition verbs
- Miscellaneous COPR verbs

The detailed descriptions of the control operator verbs follow.

Within each subcategory, the verbs are listed alphabetically.

² All LU 6.2 verbs are described, generally, in Chapter 2, “Verb Categories” on page 2-1.

Chapter 8. Change Number of Session (CNOS) Verbs



CNOS Transactions

This chapter describes a subcategory of control operator verbs consisting of verbs called the change-number-of-sessions, or CNOS, verbs. The CNOS verbs change the (LU,mode) session limit, which controls the number of LU-LU sessions per mode name that are available between two LUs for allocation to conversations. The CNOS verbs apply to both single-session and parallel-session connections. The CNOS verbs are:

CHANGE_SESSION_LIMIT
INITIALIZE_SESSION_LIMIT
PROCESS_SESSION_LIMIT
RESET_SESSION_LIMIT

Single-Session Connections

For single sessions and SNASVCMG and CPSVCMG sessions, the CNOS verbs change the (LU,mode) session limit only at the local LU. The remote LU is not involved in processing the change. The CNOS verbs that a control operator transaction program may issue for a single session or SNASVCMG or CPSVCMG session are:

INITIALIZE_SESSION_LIMIT
RESET_SESSION_LIMIT

Parallel-Session Connections

For parallel sessions, the CNOS verbs change the (LU,mode) session limit as well as other CNOS parameters of the LUs, and both LUs are involved in processing the changes. These other CNOS parameters control the minimum number of contention-winner LU-LU sessions for each LU, control which LU is responsible for selecting and deactivating LU-LU sessions when the (LU,mode) session limit is decreased or reset, and control the draining of allocation requests when the (LU,mode) session limit is reset.

CNOS Service Transaction Program

The two LUs cooperate in the execution of the CNOS verbs by means of a CNOS request and CNOS reply. The LU executing the control operator transaction program sends a CNOS request to the partner LU. The partner LU invokes an SNA service transaction program called the "CNOS service transaction program" (see Appendix D, "List of SNA Service Transaction Programs"), which causes the partner LU to process the CNOS request and send back a CNOS reply. The CNOS request and reply are sent on a basic conversation, referred to in this chapter as the "CNOS conversation." The CNOS conversation is normally allocated on an SNASVCMG session. However, if an SNASVCMG session is not active, because of session outage for example, the CNOS conversation may be allocated on another active session.

The LU that sends the CNOS request is referred to as the *source LU*; the LU that receives the CNOS request is referred to as the *target LU*. The execution of a CNOS verb by the two LUs is considered a *CNOS transaction*. The role of the LU as a source LU or target LU lasts for the duration of the CNOS transaction.

The CNOS verbs that a control operator transaction program may issue for parallel sessions are:

CHANGE_SESSION_LIMIT
INITIALIZE_SESSION_LIMIT
RESET_SESSION_LIMIT

Only a transaction program that has CNOS privilege may issue these verbs. The program is designated to have CNOS privilege when it is defined to the local LU (see the DEFINE_TP verb).

The CNOS verb that the CNOS service transaction program (at the target LU) issues for parallel sessions is:

PROCESS_SESSION_LIMIT

The detailed descriptions of the CNOS verbs follow, alphabetically by verb name.

CHANGE_SESSION_LIMIT

Changes the (LU,mode) session limit and contention-winner polarities for parallel-session connections. The verb applies to the group of sessions with the specified mode name between this (source) LU and the specified (target) LU. The new (LU,mode) session limit and contention-winner polarities are enforced until changed by a subsequent CNOS verb. As a consequence of changing the (LU,mode) session limit and contention-winner polarities, LU-LU sessions with the specified mode name may be activated or deactivated to conform to the new session limit and polarities.

(This verb is provided only by option set #501 for local support; it is in the LU 6.2 Base Set for remote support. See Table A-2.)

CHANGE_SESSION_LIMIT	Option No. [501]
Supplied Parameters:	
LU_NAME (<i>variable</i>) MODE_NAME (<i>variable</i>) LU_MODE_SESSION_LIMIT (<i>variable</i>)	
MIN_CONWINNERS_SOURCE (<i>variable</i>) MIN_CONWINNERS_TARGET (<i>variable</i>) RESPONSIBLE (<i>SOURCE</i>) (<i>TARGET</i>)	
	■ 601
	■ 602
Returned Parameters:	
RETURN_CODE (<i>variable</i>)	

Supplied Parameters:

LU_NAME specifies the name of the target LU to which the change in session limit and polarities applies. The LU name is a name that is valid as the **LU_NAME** parameter of the **ALLOCATE** verb (see "ALLOCATE" on page 7-2).

MODE_NAME specifies the mode name for which the session limit and polarities are to be changed. The specified mode name cannot be the SNA-defined mode names, **SNASVCMG** or **CPSVCMG**. See Appendix C, "Symbol String Conventions" on page C-1 for a discussion of mode name values.

LU_MODE_SESSION_LIMIT specifies the (LU,mode) session limit, that is, the maximum number of sessions to be allowed, between the source LU and target LU, for the specified mode name. Each mode has its own session limit.

CHANGE_SESSION_LIMIT

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

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

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

When the specified number is greater than 1/2 the new session limit (rounded downward), the target LU can negotiate this parameter to a number greater than or equal to 1/2 the new session limit (rounded downward) and less than the specified number. When the specified number is less than or equal to 1/2 the new session limit (rounded downward), the target LU cannot negotiate this parameter.

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

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

RESPONSIBLE specifies which LU is responsible for selecting and deactivating sessions as a result of a change that decreases the session limit or the maximum number of contention-winner sessions for the source or target LU; see note 5 for details. If no sessions need to be deactivated, this parameter is ignored.

- **SOURCE** specifies that the source LU is responsible. The target LU cannot negotiate this argument.
- **TARGET** specifies that the target LU is responsible. The target LU can negotiate this argument to SOURCE, in which case the source LU becomes responsible.

The responsible LU can deactivate a session when both LUs are finished using the session. This verb will not terminate active conversations.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- OK, with one of the following subcodes:
 - AS_SPECIFIED

- AS_NEGOTIATED
- ALLOCATION_ERROR (Refer to Appendix F, “Conversation Return Codes” on page F-1 for a complete list of subcodes with their explanations):
- COMMAND_RACE_REJECT
- LU_MODE_SESSION_LIMIT_ZERO
- LU_SESSION_LIMIT_EXCEEDED
- PARAMETER_ERROR, for one of the following reasons:
 - Invalid LU name
 - Invalid mode name
- PROGRAM_PARAMETER_CHECK, for one of the following reasons:
 - The program issuing this verb does not have CNOS privilege.
 - MODE_NAME specifies the SNA-defined mode names, SNASVCMG or CPSVCMG.
 - LU_MODE_SESSION_LIMIT specifies 0.
 - The sum of MIN_CONWINNERS_SOURCE and MIN_CONWINNERS_TARGET specifies a number greater than LU_MODE_SESSION_LIMIT.
- REQUEST_EXCEEDS_MAX_ALLOWED (Option#613 only)
- RESOURCE_FAILURE_NO_RETRY
- UNRECOGNIZED_MODE_NAME

Notes:

1. This verb applies only to parallel-session connections.
2. This verb will not terminate active conversations.
3. All of the parallel sessions between two LUs can be partitioned into groups, with all the sessions in a group having the same mode name. This verb is used to change the limits on the number of active sessions that can exist concurrently within a mode-name group between the source LU and target LU. The limits imposed on the number of active parallel sessions within a mode-name group are:
 - a. The number of active sessions cannot exceed the (LU,mode) session limit.
 - b. The number of active contention-winner sessions for the source LU cannot exceed the (LU,mode) session limit minus the minimum number of contention-winner sessions for the target LU.
 - c. The number of active contention-winner sessions for the target LU cannot exceed the (LU,mode) session limit minus the new minimum number of contention-winner sessions for the source LU.

As a result of issuing this verb, sessions may be activated, deactivated, or both to conform to the new limits. The next two notes describe the conditions under which sessions may be activated or deactivated.

4. The source or target LU may activate parallel sessions, up to the limits given in note 3, as follows:
 - It may activate new contention-winner sessions when the number of its active contention-winner sessions is less than the new (LU,mode) session

CHANGE_SESSION_LIMIT

limit minus the new minimum number of contention-winner sessions for its partner LU.

- It may activate new contention-loser sessions when the number of its active contention-loser sessions is less than the new (LU,mode) session limit minus its own new minimum number of contention-winner sessions.

The LU may activate contention-winner or -loser sessions in response to allocation requests or by means of the `ACTIVATE_SESSION` verb. Also, it may activate contention-winner sessions automatically, after completion of this verb, up to the lesser of the its new minimum number of contention-winner sessions and its automatic-activation limit currently in effect (see the `DEFINE_MODE` verb).

5. Parallel sessions are deactivated when this verb causes one or more of the limits given in note 3 to be exceeded. The LU responsible for selecting and deactivating sessions is designated by the `RESPONSIBLE` parameter. The responsible LU deactivates sessions until all three limits given in note 3 are met. When all three limits are met, no more sessions are deactivated.

The responsible LU deactivates sessions that are not allocated to conversations. If a session to be deactivated is currently allocated to a conversation, the responsible LU waits until the conversation is deallocated and then deactivates the session.

The responsible LU can deactivate only those sessions that are in excess of an LU's minimum number of contention-winner sessions. If the number of currently active contention-winner sessions for the source or target LU is less than or equal to its minimum number of contention-winner sessions, none of its contention-winner sessions are deactivated.

6. For **Intra-LU sessions**, such as those illustrated at the bottom of Figure 8-1 on page 8-7 where the local and remote LUs are the same, the parameter values for maximum number of sessions between LUs (for a particular mode name, including `SNASVCMG` and `CPSVCMG`) and the minimum number of conwinners at each LU are *always* "negotiated" in the following manner:

the maximum number of sessions specified is doubled,

the minimum number of conwinners is divided equally between source and target with each having half the maximum number of sessions, and

both values specified for minimum number of conwinners are ignored

as is shown by the two examples in Table 8-1 on page 8-7. If no errors are found, control is returned with an "OK - As Negotiated" return code. The negotiated values, not the originally specified values, are displayed by the `DISPLAY_LOCAL_LU` and `DISPLAY_MODE` verbs for intra-LU sessions. The negotiated values for intra-LU sessions represent not the number of sessions but, instead, the number of half-sessions. In the inter-LU case, the number of half-sessions is exactly equal to the number of sessions; in the intra-LU case, the number of half-sessions is exactly twice the number of sessions (because both ends of the session are in the same LU). See the half-session indicators (the little circles) at the bottom of Figure 8-1 on page 8-7.

CHANGE_SESSION_LIMIT

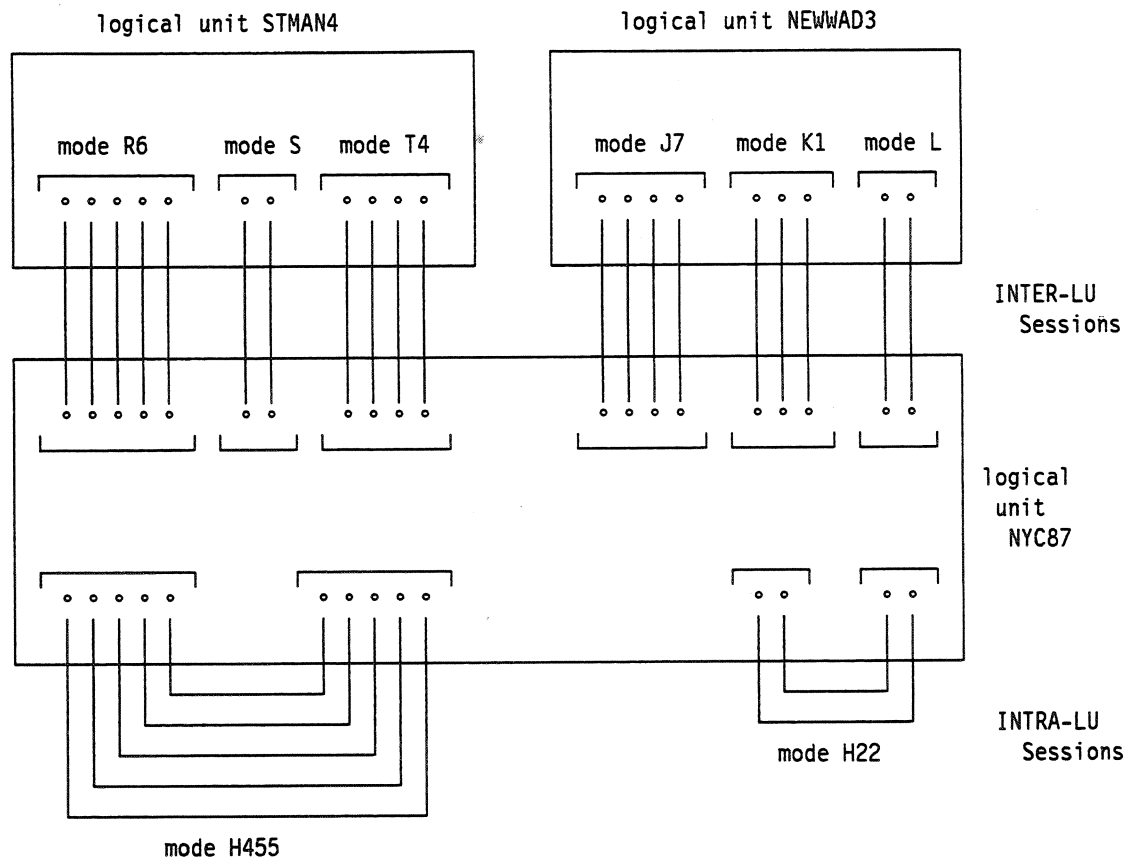


Figure 8-1. Example of Inter-LU and Intra-LU Sessions

Specified or Negotiated	LU_MODE _SESSION_LIMIT	MIN_ CONWINNERS_ SOURCE	MIN_ CONWINNERS_ TARGET
Specified	5	3	2
Negotiated	10	5	5
Specified	5	1	2
Negotiated	10	5	5

INITIALIZE_SESSION_LIMIT

INITIALIZE_SESSION_LIMIT

Establishes the initial (LU,mode) session limit for single- or parallel-session connections, and the contention-winner polarities for parallel-session connections. The verb applies to the group of sessions with the specified mode name between the source LU and the target LU. The new (LU,mode) session limit and contention-winner polarities are enforced until changed by a subsequent CNOS verb. As a consequence of initializing the session limit, one or more LU-LU sessions with the specified mode name may be activated.

INITIALIZE_SESSION_LIMIT	Option No. [Base]
Supplied Parameters:	
LU_NAME (variable) MODE_NAME (variable) ('SNASVCMG') ('CPSVCMG')	616
LU_MODE_SESSION_LIMIT (variable) MIN_CONWINNERS_SOURCE (variable)	
MIN_CONWINNERS_TARGET (variable)	601
Returned Parameters:	
RETURN_CODE (variable)	

Supplied Parameters:

LU_NAME specifies the name of the target LU to which the initialization of session limit and polarities applies. The LU name is a name that is valid as the LU_NAME parameter of the ALLOCATE verb (see "ALLOCATE" on page 7-2).

MODE_NAME specifies the mode name for which the session limit and polarities are to be initialized.

- **variable** contains the mode name.
- **'SNASVCMG'** specifies the SNA-defined mode name that is used for exchanging the CNOS request and reply when the source LU and target LU are connected by parallel sessions.
- **'CPSVCMG'** specifies the SNA-defined mode name that is used for exchanging control messages between the source and target APPN CPs (engaged in LU-LU sessions).

INITIALIZE_SESSION_LIMIT

See Appendix C, "Symbol String Conventions" for a discussion of mode name values.

LU_MODE_SESSION_LIMIT specifies the (LU,mode) session limit for parallel-session connections, that is, the maximum number of parallel sessions to be allowed, between the source LU and target LU, for the specified mode name. Each mode has its own session limit.

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

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

For single-session connections, the specified (LU,mode) session limit must be 1.

For the **SNASVCMG** or **CPSVCMG** mode name, the specified (LU,mode) session limit must be 2.

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

When the specified number is greater than 1/2 the new session limit (rounded downward), the target LU can negotiate this parameter to a number greater than or equal to 1/2 the new session limit (rounded downward) and less than the specified number. When the specified number is less than or equal to 1/2 the new session limit (rounded downward), the target LU cannot negotiate this parameter.

For single-session connections, the specified minimum number of contention-winner sessions for the source LU may be 0, or it may be 1 if the value specified on the **MIN_CONWINNER_TARGET** parameter is 0.

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

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

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

For single-session connections, the specified minimum number of contention-winner sessions for the target LU may be 0, or it may be 1 if the value specified on the **MIN_CONWINNER_SOURCE** parameter is 0.

INITIALIZE_SESSION_LIMIT

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

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- OK, with one of the following subcodes:
 - AS_SPECIFIED
 - AS_NEGOTIATED
- ALLOCATION_ERROR [Refer to Appendix F, “Conversation Return Codes” on page F-1 for a complete list of subcodes with their explanations.]
- COMMAND_REJECT
- LU_MODE_SESSION_LIMIT_CLOSED
- LU_MODE_SESSION_LIMIT_NOT_ZERO
- LU_SESSION_LIMIT_EXCEEDED
- PARAMETER_ERROR, for one of the following reasons:
 - Invalid LU name
 - Invalid mode name
- PROGRAM_PARAMETER_CHECK, for one of the following reasons:
 - The program issuing this verb does not have CNOS privilege.
 - LU_MODE_SESSION_LIMIT specifies a value greater than 1 for a single-session connection.
 - MODE_NAME('SNASVCMG') is specified and LU_MODE_SESSION_LIMIT, MIN_CONWINNERS_SOURCE, and MIN_CONWINNERS_TARGET do not specify 2, 1, and 1, respectively.
 - MODE_NAME('CPSVCMG') is specified and LU_MODE_SESSION_LIMIT, MIN_CONWINNERS_SOURCE, and MIN_CONWINNERS_TARGET do not specify 2, 1, and 1, respectively.
 - LU_MODE_SESSION_LIMIT specifies 0.
 - The sum of MIN_CONWINNERS_SOURCE and MIN_CONWINNERS_TARGET specifies a number greater than LU_MODE_SESSION_LIMIT.
- REQUEST_EXCEEDS_MAX_ALLOWED (Option 613 only)
- RESOURCE_FAILURE_NO_RETRY
- UNRECOGNIZED_MODE_NAME

Notes:

1. The (LU,mode) session limit for a single-session connection is initialized only locally at the source LU; a CNOS request and reply are not exchanged between the two LUs. Thus, the INITIALIZE_SESSION_LIMIT verb must be issued at both LUs before either LU can activate the corresponding session. From each LU's perspective, each is the source LU for the processing of this verb.
2. For a single-session connection, the contention-winner polarity for the session is determined from the MIN_CONWINNER_SOURCE and MIN_CONWINNER_TARGET parameters on the verbs issued at both LUs. The LU activating the session:

INITIALIZE_SESSION_LIMIT

- Requests that it be the contention winner when the verb issued at that LU specifies MIN_CONWINNERS_TARGET(0).
- Requests that it be the contention loser when the verb specifies MIN_CONWINNERS_TARGET(1).

The partner LU:

- Accepts the requested contention-winner polarity when the verb issued at that LU specifies MIN_CONWINNERS_SOURCE(0).
 - Negotiates the polarity so that it is the contention winner when the verb specifies MIN_CONWINNERS_SOURCE(1).
3. For a single-session connection, the LU may have more than one mode defined at a time to a partner LU. Each (LU,mode) session limit can be either 0 or 1, and more than one (LU,mode) session limit can be 1, concurrently, for the partner LU. This permits the LU to activate a session for any of the modes that have an (LU,mode) session limit of 1; however, only one session can be active at a time.
 4. For a parallel-session connection to a target LU, the (LU,mode) session limit and contention-winner polarities for the SNASVCMG or CPSVCMG mode name and target LU must be initialized before (LU,mode) session limit and contention-winner polarities can be initialized for any other mode name for the target LU. The (LU,mode) session limit and contention-winner polarities for the SNASVCMG mode name and target LU are initialized only at the source LU; a CNOS request and reply are not exchanged between the two LUs.

The (LU,mode) session limit and contention-winner polarities for the SNASVCMG or CPSVCMG mode name must be initialized at both the source LU and target LU before either LU can activate the corresponding sessions; a CNOS request and reply are not exchanged between the two LUs. From each LU's perspective, each is the source LU for the processing of INITIALIZE_SESSION_LIMIT with MODE_NAME('SNASVCMG') or MODE_NAME('CPSVCMG').

5. All the parallel sessions between two LUs can be partitioned into groups, with all the sessions in a group having the same mode name. This verb can be used to initialize the limits on the number of active parallel sessions that can exist concurrently within a mode-name group between the source and target LUs. The limits imposed on the number of active parallel sessions within a mode-name group are:
 - a. The number of active sessions cannot exceed the (LU,mode) session limit.
 - b. The number of active contention-winner sessions for the source LU cannot exceed the (LU,mode) session limit minus the minimum number of contention-winner sessions for the target LU.
 - c. The number of active contention-winner sessions for the target LU cannot exceed the (LU,mode) session limit minus the new minimum number of contention-winner sessions for the source LU.

As a result of issuing this verb, parallel sessions may be activated to conform to the new limits.

INITIALIZE_SESSION_LIMIT

6. For single- and parallel-session connections, an LU may activate contention-winner or -loser sessions in response to allocation requests or by means of the `ACTIVATE_SESSION` verb. Also, it may activate contention-winner sessions automatically, after completion of this verb, up to the lesser of the its new minimum number of contention-winner sessions and its automatic-activation limit currently in effect.
7. See Note 6 on page 8-6 regarding intra-LU sessions.

PROCESS_SESSION_LIMIT

Processes the session limit, contention-winner polarities, and related CNOS parameters from the source LU and, if necessary, negotiates them to values acceptable to the target LU, for parallel session connections only.

PROCESS_SESSION_LIMIT	<i>Option No. [Base]</i>
Supplied Parameters:	
RESOURCE (<i>variable</i>)	
Returned Parameters:	
RETURN_CODE (<i>variable</i>)	
LU_NAME (<i>variable</i>)	
MODE_NAME (<i>variable1, variable2</i>)	

Supplied Parameters:

RESOURCE specifies the resource ID of the conversation that started this program.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- OK, with one of the following subcodes:
 - AS_SPECIFIED
 - AS_NEGOTIATED
- PROGRAM_PARAMETER_CHECK, for the following reason:
 - The program issuing this verb is not the SNA service transaction program identified as hex 06F1.
- RESOURCE_FAILURE_NO_RETRY

LU_NAME specifies the variable in which is returned the name of the source LU.

MODE_NAME specifies the variables in which are returned an indication of whether one or all mode names are affected, and, if one, the specific mode name.

- **variable1** is the variable in which is returned an indication of whether one or all mode names associated with the source LU are affected.
 - ONE indicates a specific mode name is affected. The mode name is returned in **variable2**.
 - ALL indicates all mode names are affected. Nothing is placed in **variable2**.

PROCESS_SESSION_LIMIT

- **variable2** is the variable in which is returned the specific mode name when only one mode is affected.

See Appendix C, "Symbol String Conventions" for a discussion of mode name values.

Notes:

1. This verb applies only to parallel-session connections.
2. This verb is issued by an SNA service transaction program called the "CNOS service transaction program," identified with the name of hex 06F1. The CNOS service transaction program is invoked at the target LU as a result of a CNOS verb being issued at the source LU. The CNOS service transaction program then issues this verb in order to initiate the target LU's processing of the CNOS request sent by the source LU.
3. The program issues the DISPLAY_MODE verb in order to obtain the new session limit, contention-winner polarities, and related CNOS parameters.

RESET_SESSION_LIMIT

Resets to 0 the (LU,mode) session limit for both single or parallel-session connections, and the contention-winner polarities for the parallel-session connections. The verb applies to the group of sessions with the specified mode name, or all mode names, between the source LU and the target LU. The reset (LU,mode) session limit and contention-winner polarities are enforced until initialized by a subsequent INITIALIZE_SESSION_LIMIT verb. As a consequence of resetting the session limit and polarities, all active sessions with the specified mode name, or all mode names, are deactivated.

RESET_SESSION_LIMIT	Option No. [Base]
Supplied Parameters:	
LU_NAME (variable)	
MODE_NAME	
(ALL)	
(ONE (variable))	
(ONE ('SNASVCMG'))	
(ONE ('CPSVCMG'))	616
RESPONSIBLE	
(SOURCE)	
(TARGET)	602
DRAIN_SOURCE	
(NO)	
(YES)	
DRAIN_TARGET	
(YES)	
(NO)	603
FORCE	
(NO)	
(YES)	604
Returned Parameters:	
RETURN_CODE (variable)	

Supplied Parameters:

LU_NAME specifies the name of the target LU to which the resetting of the session limit and polarities applies. The LU name is a name that is valid as the LU_NAME parameter of the ALLOCATE verb (see "ALLOCATE" on page 7-2).

RESET_SESSION_LIMIT

MODE_NAME specifies the mode name for which the session limit and polarities are to be reset to 0.

- **ALL** specifies that the session limit and polarities for all mode names that apply to the target LU are to be reset to 0, except for the SNA-defined mode name, SNASVCMG or CPSVCMG, which remains unchanged.
- **ONE(variable)** specifies that the session limit and polarities for only the specified mode name are to be reset to 0.
- **ONE('SNASVCMG')** specifies that the session limit and polarities are to be reset to 0 for the SNA-defined mode name that is used for exchanging the CNOS request and reply when the source LU and target LU are connected by parallel sessions.
- **ONE('CPSVCMG')** specifies that the session limit and polarities are to be reset to 0 for the SNA-defined mode name that is used for exchanging control messages between the source LU and target LU.

See Appendix C, "Symbol String Conventions" for a discussion of mode name values.

RESPONSIBLE specifies which LU is responsible for deactivating the sessions as a result of resetting the session limit for parallel-session connections. This parameter is not applicable to single-session connections or the SNASVCMG or CPSVCMG sessions.

- **SOURCE** specifies that the source LU is responsible. The target LU cannot negotiate this argument.
- **TARGET** specifies that the target LU is responsible. The target LU can negotiate this argument to SOURCE, in which case the source LU becomes responsible.

The parameters DRAIN_SOURCE and DRAIN_TARGET determine when the responsible LU can deactivate the sessions:

- If an LU is to drain its allocation requests, it continues to allocate conversations to active sessions. The responsible LU deactivates a session only when the conversation allocated to the session is deallocated and no request is awaiting allocation to any session with the specified mode name. The allocation of an awaiting request takes precedence over the deactivation of a session.
- If an LU is not to drain its allocation requests, the responsible LU deactivates a session as soon as the conversation allocated to the session is deallocated. If no conversation is allocated to the session, the responsible LU deactivates the session immediately.

In no case, however, does this verb force deallocation of active conversations.

The RESPONSIBLE and MODE_NAME parameters are interrelated, as follows:

- If MODE_NAME(ALL) is specified, RESPONSIBLE is ignored for those mode names for which the session limit is currently 0.
- If MODE_NAME(ONE(variable)) is specified and the current session limit for that mode name is 0, RESPONSIBLE is ignored.

RESET_SESSION_LIMIT

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

- **NO** specifies that the source LU cannot drain its allocation requests. All requests currently awaiting allocation, or subsequently issued, at the source LU are rejected with a return code indicating the session limit is 0.
- **YES** specifies that the source LU can drain its allocation requests. The source LU continues to allocate conversations to the sessions until no requests are awaiting allocation, at which time its draining is ended. All allocation requests issued at the source LU after draining is ended are rejected with a return code indicating the session limit is 0.

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

- If **MODE_NAME(ALL)** and **DRAIN_SOURCE(YES)** are specified, **DRAIN_SOURCE** is ignored for those mode names for which the session limit is currently 0.
- If **MODE_NAME(ALL)** and **DRAIN_SOURCE(NO)** are specified, **DRAIN_SOURCE** is accepted for all mode names, regardless of the current session limit.
- If **MODE_NAME(ONE(variable))** is specified, the current session limit for that mode name is 0, and **DRAIN_SOURCE(YES)** is currently in effect, **DRAIN_SOURCE(NO)** if specified causes the source LU's draining to terminate.
- If **MODE_NAME(ONE(variable))** is specified, the current session limit for that mode name is 0, and **DRAIN_SOURCE(NO)** is currently in effect, **DRAIN_SOURCE(NO)** must be specified.

DRAIN_TARGET specifies whether the target LU can drain its allocation requests. This parameter is not applicable to the SNASVCMG or CPSVCMG sessions.

- **NO** specifies that the target LU cannot drain its allocation requests. All requests currently awaiting allocation, or subsequently issued, at the target LU are rejected with a return code indicating the session limit is 0. For parallel-session connections, the target LU cannot negotiate this argument.
- **YES** specifies that the target LU can drain its allocation requests. The target LU continues to allocate conversations to the sessions until no requests are awaiting allocation, at which time its draining is ended. All allocation requests issued at the target LU after draining is ended are rejected with a return code indicating the session limit is 0. For parallel-session connections, the target LU can negotiate this argument to **NO**, in which case the target LU cannot drain its allocation requests.

For parallel-session connections, the **DRAIN_TARGET** and **MODE_NAME** parameters are interrelated, as follows:

- If **MODE_NAME(ALL)** and **DRAIN_TARGET(YES)** are specified, **DRAIN_TARGET** is ignored for those mode names for which the session limit is currently 0.

RESET_SESSION_LIMIT

- If **MODE_NAME(ALL)** and **DRAIN_TARGET(NO)** are specified, **DRAIN_TARGET** is accepted for all mode names, regardless of the current session limit.
- If **MODE_NAME(ONE(variable))** is specified, the current session limit for that mode name is 0, and **DRAIN_TARGET(YES)** is currently in effect, **DRAIN_TARGET(NO)** if specified causes the target LU's draining to terminate.
- If **MODE_NAME(ONE(variable))** is specified, the current session limit for that mode name is 0, and **DRAIN_TARGET(NO)** is currently in effect, **DRAIN_TARGET(YES)** if specified can be either accepted by the target LU or negotiated to NO. If accepted, the target LU can drain its remaining allocation requests if draining has not already ended.

FORCE specifies whether the source LU is to force the resetting of its session limit when certain error conditions occur that prevent successful exchange of the CNOS request and reply. This parameter is not applicable to single-session connections or the SNASVCMG or CPSVCMG sessions.

- **NO** specifies that the session limit is to be reset only upon successful completion of the exchange of the CNOS request and reply.
- **YES** specifies that the session limit is to be reset upon either successful or unsuccessful completion of the exchange of the CNOS request and reply, except for certain error conditions (see the **RETURN_CODE** parameter). If a forced reset occurs, the source LU's session limit is reset, and **RESPONSIBLE(SOURCE)** and **DRAIN_SOURCE(NO)** are assumed (regardless of what the respective parameters specify). The target LU's CNOS parameters may not be changed, depending on the error condition and when it occurred during the CNOS exchange.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program. The **FORCE** parameter determines which of the following return codes can be returned to the program.

- If **FORCE(NO)** is specified, one of the following return codes is returned:
 - OK, with one of the following subcodes:
 - **AS_SPECIFIED**
 - **AS_NEGOTIATED**
 - **ALLOCATION_ERROR** (Refer to Appendix G, "Control-Operator Return Codes" for a complete list of subcodes with their explanations.)
 - **COMMAND_RACE_REJECT**
 - **LU_MODE_SESSION_LIMIT_CLOSED**
 - **PARAMETER_ERROR**, for one of the following reasons:
 - Invalid LU name
 - Invalid mode name
 - **PROGRAM_PARAMETER_CHECK**, for one of the following reasons:
 - The program issuing this verb does not have CNOS privilege.
 - **MODE_NAME(ONE('SNASVCMG'))** is specified and one or more (LU,mode) session limits for the target LU are not 0.

RESET_SESSION_LIMIT

- MODE_NAME(ONE('CPSVCMG')) is specified and one or more (LU,mode) session limits for the target LU are not 0.
 - MODE_NAME(ONE(variable)) and DRAIN_SOURCE(YES) are specified, the current (LU,mode) session limit is 0, and DRAIN_SOURCE(NO) is currently in effect.
 - RESOURCE_FAILURE_NO_RETRY
 - UNRECOGNIZED_MODE_NAME
- If FORCE(YES) is specified, one of the following return codes is returned:
 - OK, with one of the following subcodes:
 - AS_SPECIFIED
 - AS_NEGOTIATED
 - FORCED
 - COMMAND_REJECT
 - PARAMETER_ERROR, for one of the following reasons:
 - Invalid LU name
 - Invalid mode name
 - PROGRAM_PARAMETER_CHECK, for one of the following reasons:
 - The program issuing this verb does not have CNOS privilege.
 - MODE_NAME(ONE('SNASVCMG')) is specified and one or more non-control (LU,mode) session limits for the target LU are not 0.
 - MODE_NAME(ONE('CPSVCMG')) is specified and one or more non-control (LU,mode) session limits for the target LU are not 0.
 - MODE_NAME(ONE(variable)) and DRAIN_SOURCE(YES) are specified, the current (LU,mode) session limit is 0, and DRAIN_SOURCE(NO) is currently in effect.

Notes:

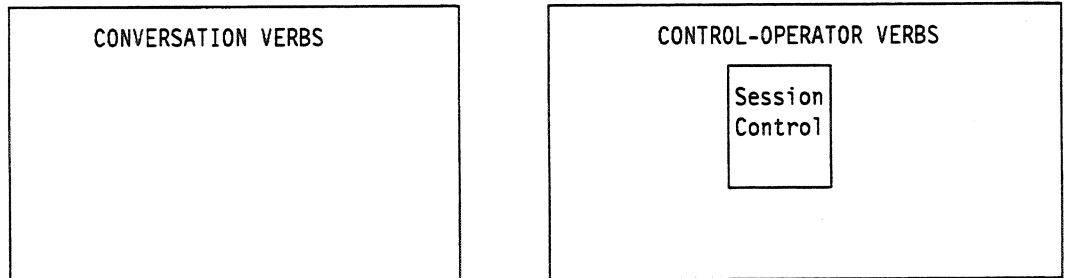
1. The (LU,mode) session limit for a single-session connection to a target LU is reset only at the source LU; a CNOS request and reply are not exchanged between the two LUs. The source LU deactivates the session, if it is active, in accordance with the DRAIN_SOURCE and DRAIN_TARGET parameters.
2. For parallel-session connections, when a mode name is specified other than the SNA-defined mode name, SNASVCMG or CPSVCMG, or when ALL mode names are indicated, the responsible LU deactivates the sessions associated with the specified mode name, or all mode names other than SNASVCMG or CPSVCMG, in accordance with the DRAIN_SOURCE and DRAIN_TARGET parameters. The (LU,mode) session limits and contention-winner polarities for all mode names other than SNASVCMG or CPSVCMG must be reset before issuing this verb with the SNASVCMG or CPSVCMG mode name specified.
3. When the SNASVCMG or CPSVCMG mode name is specified, the (LU,mode) session limit and contention-winner polarities for the SNASVCMG or CPSVCMG mode name are reset only at the source LU; a CNOS request and reply are not exchanged between the two LUs. The source LU deactivates the sessions associated with the SNASVCMG or CPSVCMG mode name as soon as all other active sessions between the source LU and target LU are deactivated. If no other sessions between the two LUs are active, the source LU

RESET_SESSION_LIMIT

immediately deactivates the sessions associated with the SNASVCMG or CPSVCMG mode name.

4. This verb can be issued when the (LU,mode) session limit is 0. In particular, this verb may be issued multiple times without issuing an intervening INITIALIZE_SESSION_LIMIT verb. For example, if this verb was first issued specifying DRAIN_TARGET(YES) and subsequently it is decided to disallow further draining by the target LU, this verb can be issued a second time specifying DRAIN_TARGET(NO). When the (LU,mode) session limit is already 0, the RESPONSIBLE parameter is ignored; the LU (SOURCE or TARGET) specified on the first RESET_SESSION_LIMIT remains responsible for deactivating sessions.

Chapter 9. Session Control Verbs



Session Control Functions

This chapter describes a subcategory of control-operator verbs consisting of verbs used for session control, activate an LU-LU session, deactivate an LU-LU session, and deactivate a session associated with a conversation group. The LU executing the verb is designated the source LU and is responsible for the session activation or deactivation. The other LU for the session is the target LU. These verbs are:

ACTIVATE_SESSION
DEACTIVATE_CONVERSATION_GROUP
DEACTIVATE_SESSION

Only a program that has session-control privilege may issue these verbs. The program is designated to have session-control privilege when it is defined to the local LU (see the DEFINE_TP verb).

The detailed descriptions of these session control verbs follow, alphabetically by verb name.

ACTIVATE_SESSION

ACTIVATE_SESSION

Activates a session with the specified mode name to the target LU. The session is activated as a contention winner for either the source LU or target LU.

ACTIVATE_SESSION	Option No. [502]
Supplied Parameters:	
LU_NAME (variable) MODE_NAME (variable) ('SNASVCMG') ('CPSVCMG')	
Returned Parameters:	
RETURN_CODE (variable)	

■ 616

Supplied Parameters:

LU_NAME specifies the name of the target LU to which the session is to be activated. This LU name is any name by which the source LU knows the target LU for the purpose of activating a session. The source LU transforms this locally known LU name to an LU name used by the network, if the names are different.

MODE_NAME specifies the mode name for the session.

- **variable** contains the mode name.
- **'SNASVCMG'** specifies the SNA-defined mode name that is used for exchanging the CNOS request and reply when the source LU and target LU are connected by parallel sessions.
- **'CPSVCMG'** specifies the SNA-defined mode name that is used for exchanging control messages between the source and target APPN CPs (engaged in LU-LU sessions).

See Appendix C, "Symbol String Conventions" for a discussion of mode name values.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- OK, with one of the following subcodes:
 - **AS_SPECIFIED** (only for single-session connection and contention winner)

ACTIVATE_SESSION

- AS_NEGOTIATED (only for single-session connection and contention winner)
- ACTIVATION_FAILURE_NO_RETRY
- ACTIVATION_FAILURE_RETRY
- PARAMETER_ERROR, for one of the following reasons:
 - Invalid LU name.
 - Invalid mode name.
- PROGRAM_PARAMETER_CHECK, for the following reason:
 - The program issuing this verb does not have session-control privilege.
- LU_MODE_SESSION_LIMIT_EXCEEDED

Notes:

1. This verb can be used to activate a single session as a contention winner for either the source LU or the target LU. The LU to be the contention winner is established by means of the INITIALIZE_SESSION_LIMIT verb.
2. This verb can be used to activate one or both parallel sessions for the SNASVCMG mode name to a target LU. The source LU is the contention winner for one of the sessions; the target LU is the contention winner for the other session.
3. This verb can be used to activate a parallel session as a contention winner for either the source LU or the target LU. The session is activated as a contention winner for the source LU when the number of currently active contention-winner sessions for the source LU is less than the new (LU,mode) session limit minus the new minimum number of contention-winner sessions for the target LU. Otherwise, the session is activated as a contention winner for the target LU.
4. The LU_LU_PASSWORD parameter, if any, is specified on the DEFINE_REMOTE_LU verb.

DEACTIVATE_CONVERSATION_GROUP

DEACTIVATE_CONVERSATION_GROUP

Deactivates the session associated with the specified conversation group ID. The type of deactivation can be cleanup or normal. See also the CONVERSATION_GROUP_ID parameter on the (MC_)ALLOCATE verb.

DEACTIVATE_CONVERSATION_GROUP	Option No. [401]
Supplied Parameters:	
CONVERSATION_GROUP_ID (variable)	
TYPE (CLEANUP) (NORMAL)	
SENSE_CODE (NO) (YES (variable))	
RETURN_CONTROL (IMMEDIATE) (DELAYED)	
108	
Returned Parameters:	
RETURN_CODE (variable)	

Supplied Parameters:

CONVERSATION_GROUP_ID specifies the identifier of the conversation group to be deactivated.

TYPE specifies the type of deactivation.

- **CLEANUP** specifies that the session is to be deactivated immediately, regardless of whether a conversation is currently associated with the session.
- **NORMAL** specifies that the session is to be deactivated normally, after all conversations associated with the session are deallocated. If no conversation is currently associated with the session, normal deactivation begins immediately.

SENSE_CODE specifies a sense data value to be used in the session deactivation processing. This parameter is significant only when TYPE(CLEANUP) is also specified. It is ignored otherwise.

- **NO** specifies that a sense data value is not present.
- **YES** specifies that a sense data value is present.

DEACTIVATE_CONVERSATION_GROUP

- Variable contains the sense data value to be used in the session deactivation processing. See *SNA Formats* for the allowable sense data values.

RETURN_CONTROL specifies when control should be returned to the transaction program issuing the verb.

- **IMMEDIATE** specifies that control should be returned immediately, without waiting for the session deactivation to complete.
- **DELAYED** specifies that control should not be returned until the session deactivation is complete.

Note: Data from the deactivated session to existing conversations may still be delivered but no new conversations can be started.

RETURN_CONTROL(DELAYED) should be used only with **TYPE(CLEANUP)** to avoid delays in returning control.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- OK
- **PROGRAM_PARAMETER_CHECK**, for one of the following reasons:
 - This verb is not supported.
 - The program issuing this verb does not have session-control privilege.

Notes:

1. Each session is assigned a unique conversation group ID when it is activated. The conversation group ID exists for the duration of the session. When the session is deactivated (normally or otherwise), the conversation group ID for the session is deleted and no longer exists.
2. If the **CONVERSATION_GROUP_ID** parameter specifies a value that does not match an existing conversation group ID, the session is considered to have been previously deactivated (or never activated), and a return code of OK is returned to the program. That is, a return code of OK means the session will be, or has been, deactivated, or it was never active. No error check is performed on the **CONVERSATION_GROUP_ID** value for validity.
3. The session deactivation process may begin during or after execution of this verb. The later case can occur, for example, when **TYPE(NORMAL)** is specified and a conversation is currently active on the session.
4. No error check is performed on the **SENSE_CODE** value for validity. (See *SNA Formats* for valid sense data values.)

DEACTIVATE_SESSION

DEACTIVATE_SESSION

Deactivates the specified LU-LU session. The type of deactivation can be cleanup or normal.

DEACTIVATE_SESSION	Option No. [504]
Supplied Parameters:	
SESSION_ID (<i>variable</i>)	
TYPE (<i>CLEANUP</i>) (<i>NORMAL</i>)	
SENSE_CODE (<i>NO</i>) (<i>YES</i> (<i>variable</i>))	
RETURN_CONTROL (<i>IMMEDIATE</i>) (<i>DELAYED</i>)	
108	
Returned Parameters:	
RETURN_CODE (<i>variable</i>)	

Supplied Parameters:

SESSION_ID specifies the identifier of the LU-LU session to be deactivated.

TYPE specifies the type of deactivation.

- **CLEANUP** specifies that the session is to be deactivated immediately, regardless of whether a conversation is currently allocated to the session.
- **NORMAL** specifies that the session is to be deactivated normally, after the conversation currently allocated to the session is deallocated. If no conversation is currently allocated to the session, normal deactivation begins immediately.

SENSE_CODE specifies a sense data value to be used in the session deactivation processing. This parameter is significant only when **TYPE(CLEANUP)** is also specified. It is ignored otherwise.

- **NO** specifies that a sense data value is not present.
- **YES** specifies that a sense data value is present.
 - Variable contains the sense data value to be used in the session deactivation processing. See *SNA Formats* for the allowable sense data value values.

DEACTIVATE_SESSION

RETURN_CONTROL specifies when control should be returned to the transaction program issuing the verb.

- **IMMEDIATE** specifies that control should be returned immediately, without waiting for the session deactivation to complete.
- **DELAYED** specifies that control should not be returned until the session deactivation is complete.

Note: Data from the deactivated session to existing conversations may still be delivered but no new conversations can be started.

RETURN_CONTROL(DELAYED) should be used only with **TYPE(CLEANUP)** to avoid delays in returning control.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

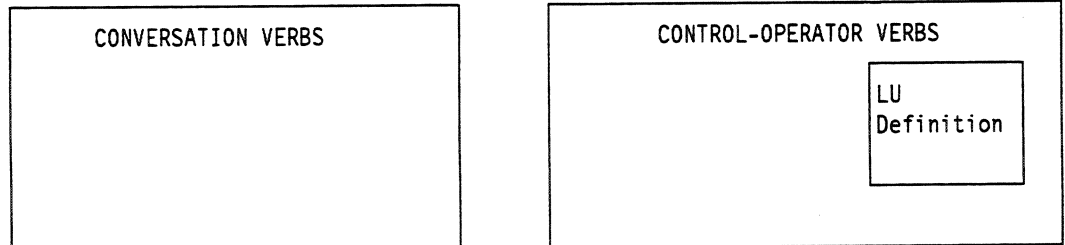
- OK
- **PROGRAM_PARAMETER_CHECK**, for the following reason:
 - The program issuing this verb does not have session-control privilege.

Notes:

1. Each session is assigned a unique session ID when it is activated. The session ID exists for the duration of the session. When the session is deactivated (normally or otherwise), the session ID for the session is deleted and no longer exists.
2. If the **SESSION_ID** parameter specifies a value that does not match an existing session ID, the session is considered to have been previously deactivated (or never activated), and a return code of OK is returned to the program. That is, a return code of OK means the session will be, or has been, deactivated, or it was never active. No error check is performed on the **SESSION_ID** value for validity.
3. The session deactivation process may begin during or after execution of this verb. The later case can occur, for example, when **TYPE(NORMAL)** is specified and a conversation is currently active on the session.
4. No error check is performed on the **SENSE_CODE** value for validity. (See *SNA Formats* for valid sense data values.)

DEACTIVATE_SESSION

Chapter 10. LU Definition Verbs



LU Definition Functions

This chapter describes a subcategory of control-operator verbs consisting of the following verbs, which are used to define or modify the local LU's operating parameters, examine the parameters, and delete the parameters. These verbs are:

DEFINE_LOCAL_LU
DEFINE_MODE
DEFINE_REMOTE_LU
DEFINE_TP

DELETE

DISPLAY_LOCAL_LU
DISPLAY_MODE
DISPLAY_REMOTE_LU
DISPLAY_TP

The execution of these verbs involves only the local LU. They do not cause any information to be sent outside the LU.

Some of the local LU's operating parameters can be added, modified, or deleted only under appropriate conditions. An attempt to alter any of these parameters when conditions are inappropriate is an error, causing the LU to return the `PARAMETER_ERROR` return code on the verb. The parameters that are restricted in this way and the corresponding errors are identified in the verb descriptions.

The `DEFINE` verbs may be issued multiple times to initialize or update the local LU's operating parameters. The first time a verb parameter is specified, the LU's corresponding operating parameter is initialized; thereafter, it is changed. The following notes apply to all verb parameters, except where stated otherwise in the individual verb descriptions:

LU Definition Verbs

- If the LU's operating parameter is not currently defined and the corresponding verb parameter is specified, the operating parameter is initialized with the supplied value.
- If the LU's operating parameter is not currently defined and the corresponding verb parameter is omitted, the operating parameter remains undefined.
- If the LU's operating parameter is currently defined and the corresponding verb parameter is specified, the operating parameter value is replaced with the supplied value.
- If the LU's operating parameter is currently defined and the corresponding verb parameter is omitted, the operating parameter value remains unchanged.

The DELETE verb deletes the local LU's operating parameters. After a parameter is deleted, it is no longer defined at the local LU.

The DISPLAY verbs return current values of the local LU's operating parameters. When a DISPLAY verb is issued specifying a parameter that is not currently defined at the local LU, a null value is returned.

Only a transaction program that has define privilege may issue the DEFINE verbs and DELETE verb, and only a program that has display privilege may issue the DISPLAY verbs. The program is designated to have define or display privilege when it is defined to the local LU (see the DEFINE_TP verb). The program that initially establishes define privilege for other programs has implicit define privilege.

The detailed descriptions of the LU definition verbs follow, alphabetically by verb name.

DEFINE_LOCAL_LU

Defines the fully qualified name for the local LU, and initializes or changes parameters that control the operation of the local LU.

DEFINE_LOCAL_LU	Option No. [505]
Supplied Parameters:	
FULLY_QUALIFIED_LU_NAME (<i>variable</i>)	
LU_SESSION_LIMIT	
(<i>NONE</i>)	
(<i>VALUE</i> (<i>variable</i>))	605
SECURITY	
(<i>ADD</i> (<i>USER_ID</i> (<i>variable</i>)))	212
(<i>PASSWORD</i> (<i>variable</i>))	212
(<i>PROFILE</i> (<i>variable</i>)))	215
(<i>DELETE</i> (<i>USER_ID</i> (<i>variable</i>)))	212
(<i>PROFILE</i> (<i>variable</i>)))	215
MAP_NAME	
(<i>ADD</i> (<i>variable</i>))	246
(<i>DELETE</i> (<i>variable</i>))	246
BIND_RSP_QUEUE_CAPABILITY	
(<i>YES</i>)	
(<i>NO</i>)	
LENGTH_CHECKED_COMPRESSION	618
(<i>PROHIBITED</i>)	618
(<i>REQUIRED</i>)	618
(<i>NEGOTIABLE</i>)	618
INBOUND_COMPRESSION_LEVEL	618
(<i>NONE</i>)	618
(<i>RLE</i>)	618
(<i>LZ9</i>)	618
(<i>LZ10</i>)	618
(<i>LZ12</i>)	618
OUTBOUND_COMPRESSION_LEVEL	618
(<i>NONE</i>)	618
(<i>RLE</i>)	618
(<i>LZ9</i>)	618
(<i>LZ10</i>)	618
(<i>LZ12</i>)	618
RLE_BEFORE_LZ	618
(<i>YES</i>)	618
(<i>NO</i>)	618

DEFINE_LOCAL_LU

Returned Parameters:

RETURN_CODE (*variable*)

Supplied Parameters:

FULLY_QUALIFIED_LU_NAME specifies the fully (or *network-*) qualified name of the local LU. If the specified name is not currently defined, this verb defines the name by which the local LU is known throughout the network, replacing the current name if one exists, and initializes the LU-LU session limit. If the specified name is already defined, this verb changes the other parameter values.

LU_SESSION_LIMIT specifies the LU-LU session limit for the total number of sessions for the local LU.

- **NONE** specifies that no limit is to be defined.
- **VALUE** specifies a number representing the LU-LU session limit.

SECURITY specifies to add or delete access security information that the local LU uses for conversation-level security verification of incoming allocation requests on an LU-wide basis. (Contrast this parameter with the **SECURITY_REQUIRED** and **SECURITY_ACCESS** parameters of the **DEFINE_TP** verb.) The local LU updates a conversation-level security verification list from the information supplied on this parameter. The verification list consists of one or more user IDs and corresponding passwords, and zero or more profiles associated with each user ID.

- **ADD** specifies to add access security information to the LU's conversation-level security verification list. A user ID must be specified together with either, or both, a password or profile.
 - **USER_ID** specifies a user ID. If the user ID is not currently defined, a password must also be specified, and the LU adds the user ID, password, and profile (if specified) to its conversation-level security verification list. If the user ID is already defined, the list is updated with the password or profile.
 - **PASSWORD** specifies the password for this user ID. If the user ID is already defined, the password replaces the one currently defined.
 - **PROFILE** specifies a profile for this user ID. If the user ID is already defined, the profile is added to the ones currently defined.
- **DELETE** specifies to delete access security information from the LU's conversation-level security verification list. The user ID may be specified alone or together with a profile.
 - **USER_ID** specifies the user ID. If a profile is not specified, the user ID and its associated password and profiles are deleted. If a profile is also specified, the user ID and its associated password and other profiles remain defined.
 - **PROFILE** specifies the profile to be deleted.

MAP_NAME specifies to add or delete a map name that the local LU is to support for local data mapping. Local transaction programs may specify this map name on the **MAP_NAME** parameter of the **MC_SEND_DATA** verb, and remote LUs may send this map name to the local LU over mapped conversations.

- **ADD** specifies the map name to be added.
- **DELETE** specifies the map name to be deleted.

BIND_RSP_QUEUE_CAPABILITY specifies whether the local LU is willing to allow the remote LU to queue session-activation (**BIND**) requests if a session cannot be activated immediately. The local LU uses this parameter to determine the indication for **BIND** response queue capability that it specifies in session-activation requests.

- **YES** specifies that the local LU is willing to allow the remote LU to queue session-activation requests.
- **NO** specifies that the local LU is not willing to allow the remote LU to queue session-activation requests.

LENGTH_CHECKED_COMPRESSION specifies whether the local LU should use length-checked compression on its sessions. The local LU uses this parameter, the **INBOUND_COMPRESSION_LEVEL** parameter, and the **OUTBOUND_COMPRESSION_LEVEL** parameter to determine the compression support that it negotiates via session-activation (**BIND**) requests and responses.

- **PROHIBITED** specifies that no compression is to be used.
- **REQUIRED** specifies that compression is required on the session.
- **NEGOTIABLE** specifies that the usage of compression is to be negotiated by the PLU and the SLU. If compression is to be used, the level of compression will be negotiated.

INBOUND_COMPRESSION_LEVEL specifies the maximum level of compression supported for inbound data.

Run-length encoding (RLE) substitutes a 1- or 2-byte sequence in the data stream for each repeated run of the same character. *Lempel-Ziv-like compression*¹, called *LZ compression* in this book, substitutes 9-, 10-, or 12-bit codes for repeated substrings in the data stream; these codes are indices that refer to entries in a common dictionary generated adaptively at both sender and receiver as the data flows and compression occurs. LZ algorithms (referred to as *LZ9*, *LZ10*, or *LZ12* according to the number of bits in the substituted codes) vary in the size of dictionaries allowed. Dictionary size influences the compression ratio (uncompressed RU size divided by compressed RU size). Typically, RLE requires less processor overhead and storage than the LZ algorithms; however, the LZ algorithms achieve better compression ratios than RLE. For the LZ algorithms, the processor overhead is about the same; but with respect to storage and compression ratios, LZ9 requires the least amount of storage but gets the smallest compression ratio; LZ12 requires the greatest amount of storage and gets the best compression ratio; LZ10 is a compromise between LZ9 and LZ12.

¹ This technique is an adaptive dictionary-based compression algorithm similar to Lempel-Ziv.

DEFINE_LOCAL_LU

A value of *NONE* indicates that no compression is to be used on inbound data for sessions involving the local LU; the other values (RLE, LZ9, LZ10, LZ12) indicate the algorithm to be used for compression.

OUTBOUND_COMPRESSION_LEVEL specifies the maximum level of compression supported for outbound data. A value of *NONE* indicates that no compression is to be used on outbound data for sessions involving the local LU; the other values indicate the algorithms to be used for compression.

RLE_BEFORE_LZ specifies that run-length encoding (RLE) is to be applied first to the data, then LZ compression to that result. This parameter applies only if LZ9, LZ10, or LZ12 compression is used on sessions involving the local LU; otherwise, LZ compression alone is used. Support of RLE before LZ compression is implementation-dependent.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- OK
- **PARAMETER_ERROR**, for one of the following reasons:
 - **FULLY_QUALIFIED_LU_NAME** specifies a value that is not a type-1134 symbol string.
 - **LU_SESSION_LIMIT(VALUE(variable))** specifies a value that is less than the sum of the (LU,mode) session limits currently in effect.
 - **SECURITY(ADD(...))** specifies a user ID, password, or profile that is not a symbol-string type (A, AE, GR, or DB) that the product supports.
 - **SECURITY(ADD(...))** specifies only a user ID, or a password or profile but no user ID.
 - **SECURITY(DELETE(...))** specifies a user ID or profile that is not currently defined at the local LU.
 - **SECURITY(DELETE(...))** specifies only a profile.
 - **MAP_NAME(ADD(variable))** specifies a map name that is not a symbol-string type (A, AE, or GR) that the product supports.
 - **MAP_NAME(DELETE(variable))** specifies a map name that is not currently defined at the local LU.
 - **INBOUND_COMPRESSION_LEVEL** specifies a compression level that is not supported.
 - **OUTBOUND_COMPRESSION_LEVEL** specifies a compression level that is not supported.
- **PROGRAM_PARAMETER_CHECK**, for the following reason:
 - The program issuing this verb does not have define privilege.

Notes:

1. This verb can be used to define the name by which the local LU is known throughout the network. To use it for this purpose, the verb should be issued

prior to the local LU's participation in any network activity, such as initializing (LU,mode) session limits.

2. The LU-LU session limit is the maximum number of sessions that the local LU can have active at a time. It represents the upper bound on the sum of the (LU,mode) session limits, and it must be equal to or greater than the sum of the (LU,mode) session limits currently in effect; see the description of the return code, LU_SESSION_LIMIT_EXCEEDED, in "Purpose of Control Operator Return Code" on page G-1 for more details.
3. If no LU-LU session limit is defined at the local LU, the upper bound, if any, on the sum of the (LU,mode) session limits is product-determined. For example, the upper bound may be a fixed value or determined by an algorithm.
4. When the first user ID and associated password and profiles are added, the LU's conversation-level security verification list is created. When the last user ID and associated password and profiles are deleted, the conversation-level security verification list itself is deleted.
5. The local LU uses the conversation-level security verification list to verify the access security information on allocation requests it receives. Specifically, when the LU receives an allocation request carrying a user ID and password, it verifies that the user ID and password are present in its conversation-level security verification list. If the allocation request also carries a profile, the LU verifies that the profile is also present in the list. Allocation requests that carry no access security information, or that carry a user ID (and optional profile) and an already-verified indication, are not verified against the conversation-level security verification list. However, these requests may be subject to resource-access verification, as determined by the SECURITY_ACCESS parameter on DEFINE_TP.
6. If the conversation-level security verification list does not exist, the local LU will perform no conversation-level security verification. Allocation requests that carry a user ID and password will not be accepted.
7. If no map names are defined at the local LU, it will perform no data mapping.
8. More details concerning the LU's use of these operating parameters are given in *SNA LU 6.2 Reference: Peer Protocols*.
9. The length-checked compression parameters

LENGTH_CHECKED_COMPRESSION
INBOUND_COMPRESSION_LEVEL
OUTBOUND_COMPRESSION_LEVEL
RLE_BEFORE_LZ

apply only to sessions in which the local LU is the PLU. (The SLU uses other means, such as outgoing link capability and the parameters on the DEFINE_MODE verb, to determine support for length-checked compression, outbound and inbound compression levels, and RLE before LZ.)

10. If length-checked compression parameters
- LENGTH_CHECKED_COMPRESSION
INBOUND_COMPRESSION_LEVEL
OUTBOUND_COMPRESSION_LEVEL

DEFINE_LOCAL_LU

RLE_BEFORE_LZ

are used on both the DEFINE_LOCAL_LU and the DEFINE_MODE verbs, the parameters on the DEFINE_MODE verb take precedence on a session involving both that LU and mode name.

11. Which length-checked compression algorithms are supported is implementation-dependent. However, an implementation that supports a Lempel-Ziv algorithm (LZ9, LZ10, or LZ12) always supports RLE. Also, an implementation that supports LZ10 always supports LZ9; likewise, an implementation that supports LZ12 always supports both LZ10 and LZ9.

DEFINE_MODE

Initializes or changes parameters that control the operation of the local LU in conjunction with a group of sessions to the specified remote LU, the session group being identified by a mode name.

DEFINE_MODE	Option No. [505]
Supplied Parameters:	
FULLY_QUALIFIED_LU_NAME (variable)	
MODE_NAME (variable)	
SEND_MAX_RU_SIZE_LOWER_BOUND (variable)	610
SEND_MAX_RU_SIZE_UPPER_BOUND (variable)	610
RECEIVE_MAX_RU_SIZE_LOWER_BOUND (variable)	610
RECEIVE_MAX_RU_SIZE_UPPER_BOUND (variable)	610
SINGLE_SESSION_REINITIATION	
(OPERATOR)	
(PLU)	608
(SLU)	608
(PLU_OR_SLU)	608
SESSION_LEVEL_CRYPTOGRAPHY	
(NOT_SUPPORTED)	
(MANDATORY)	611
(SELECTIVE)	611, 617
LENGTH_CHECKED_COMPRESSION	618
(PROHIBITED)	618
(REQUIRED)	618
(NEGOTIABLE)	618
INBOUND_COMPRESSION_LEVEL	618
(NONE)	618
(RLE)	618
(LZ9)	618
(LZ10)	618
(LZ12)	618
OUTBOUND_COMPRESSION_LEVEL	618
(NONE)	618
(RLE)	618
(LZ9)	618
(LZ10)	618
(LZ12)	618
RLE_BEFORE_LZ	618
(YES)	618
(NO)	618

DEFINE_MODE

CONWINNER_AUTO_ACTIVATE_LIMIT (<i>variable</i>)	612
PREFERRED_RECEIVE_RU_SIZE (<i>variable</i>)	610
PREFERRED_SEND_RU_SIZE (<i>variable</i>)	610
SESSION_DEACTIVATED_TP_NAME (<i>variable</i>)	401
LOCAL_MAX_SESSION_LIMIT (<i>variable</i>)	613
Returned Parameters:	
RETURN_CODE (<i>variable</i>)	

Supplied Parameters:

FULLY_QUALIFIED_LU_NAME specifies the fully (or *network-*) qualified name of the remote LU to which the other parameters of this verb apply. It is this name that is specified in the **LU_NAME** parameter of the **MC_ALLOCATE** or **ALLOCATE** verb except when option #606 or option #607 is implemented. Option #606 provides a locally known LU name; option #607 provides an uninterpreted LU name. (See Appendix A, "Base and Option Sets for Product Support.")

MODE_NAME specifies the mode name for the group of sessions to which the remaining parameters of this verb apply. If the specified name is currently undefined at the local LU for the remote LU, this verb defines the mode name and initializes the other parameter values specified on this verb. If the specified name is already defined at the local LU for the remote LU, this verb changes the other parameter values.

SEND_MAX_RU_SIZE_LOWER_BOUND specifies the lower bound for the maximum size of normal-flow requests that the local LU sends on the sessions. This value must be less than or equal to the value specified on **SEND_MAX_RU_SIZE_UPPER_BOUND**. The local LU uses these lower- and upper-bound values to determine the value for its send maximum RU size that it specifies in session-activation (**BIND**) requests and responses.

SEND_MAX_RU_SIZE_UPPER_BOUND specifies the upper bound for the maximum size of normal-flow requests that the local LU sends.

RECEIVE_MAX_RU_SIZE_LOWER_BOUND specifies the lower bound for the maximum size of normal-flow requests that the local LU receives on the sessions. This value must be less than or equal to the value specified on **RECEIVE_MAX_RU_SIZE_UPPER_BOUND**. The local LU uses these lower- and upper-bound values to determine the value for its receive maximum RU size that it specifies in session-activation (**BIND**) requests and responses.

RECEIVE_MAX_RU_SIZE_UPPER_BOUND specifies the upper bound for the maximum size of normal-flow requests that the local LU receives.

SINGLE_SESSION_REINITIATION specifies the responsibility for session reinitiation of a single session with the remote LU. The local LU uses this parameter to determine the indication for session reinitiation responsibility that it specifies in session-activation (**BIND**) requests and responses. The remote LU must be defined

to not support parallel sessions (see the `PARALLEL_SESSION_SUPPORT` parameter on the `DEFINE_REMOTE_LU` verb).

Note: This parameter is ignored on parallel-session connections.

- **OPERATOR** specifies that neither LU will automatically attempt to reinitiate the session. If a reinitiation race occurs, where the operators at both LUs attempt to reinitiate the session at the same time, the reinitiation is successfully completed by the LU with the greater fully qualified LU name (provided no session-activation errors are encountered). The comparison of the fully qualified LU names is based on their hexadecimal values.
- **PLU** specifies that the primary LU will automatically attempt to reinitiate the session.
- **SLU** specifies that the secondary LU will automatically attempt to reinitiate the session.
- **PLU_OR_SLU** specifies that either LU may automatically attempt to reinitiate the session. A reinitiation race between the two LUs is resolved in the same way as for operator reinitiation.

SESSION_LEVEL_CRYPTOGRAPHY specifies whether the local LU supports session-level cryptography for the sessions. The local LU uses this parameter to determine the indication for cryptography support that it specifies in session-activation (BIND) requests and responses.

- **NOT_SUPPORTED** specifies that no session-level cryptography is to be used.
- **MANDATORY** specifies that session-level cryptography is to be used on all FMD requests flowing on the sessions.
- **SELECTIVE** specifies that session-level cryptography is required just on selected FMD requests flowing on the sessions.

LENGTH_CHECKED_COMPRESSION specifies whether the local LU should use length-checked compression on its sessions. The local LU uses this parameter, the `INBOUND_COMPRESSION_LEVEL` parameter, and the `OUTBOUND_COMPRESSION_LEVEL` parameter to determine the compression support that it negotiates via session-activation (BIND) requests and responses.

- **PROHIBITED** specifies that no compression is to be used.
- **REQUIRED** specifies that compression is required on the session.
- **NEGOTIABLE** specifies that the usage of compression is to be negotiated by the PLU and the SLU. If compression is to be used, the level of compression will be negotiated.

INBOUND_COMPRESSION_LEVEL specifies the maximum level of compression supported for inbound data.

Run-length encoding (RLE) substitutes a 1- or 2-byte sequence in the data stream for each repeated run of the same character. *Lempel-Ziv-like compression*², called *LZ compression* in this book, substitutes 9-, 10-, or 12-bit codes for repeated sub-

² An adaptive dictionary-based compression algorithm similar to Lempel-Ziv.

DEFINE_MODE

strings in the data stream; these codes are indices that refer to entries in a common dictionary generated adaptively at both sender and receiver as the data flows and compression occurs. LZ algorithms (referred to as *LZ9*, *LZ10*, or *LZ12* according to the number of bits in the substituted codes) vary in the size of dictionaries allowed. Dictionary size influences the compression ratio (uncompressed RU size divided by compressed RU size). Typically, RLE requires less processor overhead and storage than the LZ algorithms; however, the LZ algorithms achieve better compression ratios than RLE. For the LZ algorithms, the processor overhead is about the same; but with respect to storage and compression ratios, LZ9 requires the least amount of storage but gets the smallest compression ratio; LZ12 requires the greatest amount of storage and gets the best compression ratio; LZ10 is a compromise between LZ9 and LZ12.

A value of *NONE* indicates that no compression is to be used on inbound data for sessions involving this mode; the other values (RLE, LZ9, LZ10, LZ12) indicate the algorithm to be used for compression.

OUTBOUND_COMPRESSION_LEVEL specifies the maximum level of compression supported for outbound data. A value of *NONE* indicates that no compression is to be used on outbound data for sessions involving the local LU; the other values indicate the algorithms to be used for compression.

RLE_BEFORE_LZ specifies that run-length encoding (RLE) is to be applied first to the data, then LZ compression to that result. This parameter applies only if LZ9, LZ10, or LZ12 compression is used on sessions involving the local LU; otherwise, LZ compression alone is used. Support of RLE before LZ compression is implementation-dependent.

CONWINNER_AUTO_ACTIVATE_LIMIT specifies the automatic-activation limit on the number of contention-winner sessions that the local LU can automatically activate when the minimum number of contention-winner sessions for the local LU increases (as a result of CNOS processing). The actual limit on the number of contention-winner sessions automatically activated is the lesser of the value specified on this parameter and the new minimum number of contention-winner sessions for the local LU.

A value of 0 specifies that the local LU is to automatically activate no sessions.

PREFERRED_RECEIVE_RU_SIZE specifies the preferred maximum size of normal-flow requests that the local LU receives on the sessions. This value must be less than or equal to the value specified on

RECEIVE_MAX_RU_SIZE_UPPER_BOUND and greater than or equal to the value specified on the **RECEIVE_MAX_RU_SIZE_LOWER_BOUND**. The local LU uses this value to determine the value for its receive maximum RU size that it specifies in session-activation (BIND) requests and responses.

PREFERRED_RECEIVE_RU_SIZE specifies the receive RU size preferred by this LU. However, the LU will allow its maximum receive RU size to be negotiated up to the specified **RECEIVE_MAX_RU_SIZE_UPPER_BOUND** or down to the **RECEIVE_MAX_RU_SIZE_LOWER_BOUND**.

PREFERRED_SEND_RU_SIZE specifies the preferred maximum size of normal-flow requests that the local LU sends on the sessions. This value must be less than or equal to the value specified on **SEND_MAX_RU_SIZE_UPPER_BOUND** and greater than or equal to the value specified on the **SEND_MAX_RU_SIZE_LOWER_BOUND**. The local LU uses this value to

determine the value for its send maximum RU size that it specifies in session-activation (BIND) requests and responses. **PREFERRED_SEND_RU_SIZE** specifies the send RU size preferred by this LU. However, the LU will allow its maximum send RU size to be negotiated up to the specified **SEND_MAX_RU_SIZE_UPPER_BOUND** or down to the **SEND_MAX_RU_SIZE_LOWER_BOUND**.

SESSION_DEACTIVATED_TP_NAME specifies the name of the transaction program to be invoked when a session to the LU specified on **FULLY_QUALIFIED_LU_NAME** having the mode name specified on **MODE_NAME** is deactivated for any reason. If the value specified on **SESSION_DEACTIVATED_TP_NAME** is null, no transaction program is invoked when a session is deactivated.

LOCAL_MAX_SESSION_LIMIT specifies the maximum value that the local LU is to use, during CNOS processing, for the (LU,mode) session limit. The local LU, as a target LU, will negotiate a higher session limit it receives in the CNOS request down to this maximum value. The local LU, as a source LU, will restrict the session limit it sends in a CNOS request to a value less than or equal to this maximum value.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- OK
- **PARAMETER_ERROR**, for one of the following reasons:
 - **FULLY_QUALIFIED_LU_NAME** does not specify a remote LU name defined at the local LU.
 - **FULLY_QUALIFIED_LU_NAME** specifies a value that is not a type-1134 (non-blank) symbol string.
 - **MODE_NAME** specifies a value that is not a type-1134 (non-blank) symbol string.
 - **PREFERRED_SEND_RU_SIZE** specifies a value exceeding that on **SEND_MAX_RU_SIZE_UPPER_BOUND**.
 - **PREFERRED_RECEIVE_RU_SIZE** specifies a value exceeding that on **RECEIVE_MAX_RU_SIZE_UPPER_BOUND**.
 - **PREFERRED_SEND_RU_SIZE** specifies a value less than that on **SEND_MAX_RU_SIZE_LOWER_BOUND**.
 - **PREFERRED_RECEIVE_RU_SIZE** specifies a value less than that on **RECEIVE_MAX_RU_SIZE_LOWER_BOUND**.
 - **SEND_MAX_RU_SIZE_LOWER_BOUND** specifies a value exceeding that on **SEND_MAX_RU_SIZE_UPPER_BOUND**.
 - **RECEIVE_MAX_RU_SIZE_LOWER_BOUND** specifies a value exceeding that on **RECEIVE_MAX_RU_SIZE_UPPER_BOUND**.
 - **SINGLE_SESSION_REINITIATION** is specified for a remote LU that is currently defined as supporting parallel sessions.

Note: This reason applies only to single-session connections.

DEFINE_MODE

- A parameter other than CONWINNER_AUTO_ACTIVATE_LIMIT is specified and the (LU,mode) session limit and count for the mode name are not zero.
- INBOUND_COMPRESSION_LEVEL specifies a compression level that is not supported.
- OUTBOUND_COMPRESSION_LEVEL specifies a compression level that is not supported.
- PROGRAM_PARAMETER_CHECK, for the following reason:
 - The program issuing this verb does not have define privilege.

Notes:

1. If the mode name is currently defined, the (LU,mode) session limit and count must be zero when this verb is issued specifying any of the parameters other than CONWINNER_AUTO_ACTIVATE_LIMIT. The auto-activation limit on the number of contention-winner sessions may be initialized or changed at anytime.
2. If no send or receive pacing window size is defined, a product-determined window size is used.
3. If no lower bound is defined for the send or receive maximum size of normal-flow requests, a product-determined lower bound is used.
4. If no upper bound is defined for the send or receive maximum size of normal-flow requests, a product-determined upper bound is used.
5. If no responsibility for session reinitiation of a single session is defined for the mode name, a product-determined responsibility is used.
6. If no session-level cryptography is defined for the mode name, none is used.
7. If no automatic-activation limit for contention-winner sessions is defined for the mode name, the local LU may not automatically activate any sessions, or it may automatically activate sessions up to the minimum number of the LU's contention-winner sessions currently in effect, depending on the product.
8. More details concerning the LU's use of these operating parameters are given in *SNA LU 6.2 Reference: Peer Protocols*.
9. If length-checked compression parameters
 LENGTH_CHECKED_COMPRESSION
 INBOUND_COMPRESSION_LEVEL
 OUTBOUND_COMPRESSION_LEVEL
 RLE_BEFORE_LZ
are used on both the DEFINE_LOCAL_LU and the DEFINE_MODE verbs, the parameters on the DEFINE_MODE verb take precedence on a session involving both that LU and mode name.
10. Which length-checked compression algorithms are supported is implementation-dependent. However, an implementation that supports a Lempel-Ziv algorithm (LZ9, LZ10, or LZ12) always supports RLE. Also, an implementation that supports LZ10 always supports LZ9; likewise, an implementation that supports LZ12 always supports both LZ10 and LZ9.

DEFINE_REMOTE_LU

Initializes or changes parameters that control the operation of the local LU in conjunction with a remote LU.

DEFINE_REMOTE_LU		Option No. [505]
Supplied Parameters:		
FULLY_QUALIFIED_LU_NAME (variable)		
LOCALLY_KNOWN_LU_NAME	(NONE)	
	(NAME (variable))	606
UNINTERPRETED_LU_NAME	(NONE)	
	(NAME (variable))	607
INITIATE_TYPE	(INITIATE_ONLY)	
	(INITIATE_OR_QUEUE)	
PARALLEL_SESSION_SUPPORT	(YES)	
	(NO)	
CNOS_SUPPORT	(YES)	
	(NO)	
LU_LU_PASSWORD	(NONE)	
	(VALUE (variable))	211
SECURITY_ACCEPTANCE	(NONE)	
	(CONVERSATION)	212
	(ALREADY_VERIFIED)	212
	(PERSISTENT_VERIFICATION)	220
	(AV_AND_PV)	212, 220
Returned Parameters:		
RETURN_CODE (variable)		

Supplied Parameters:

DEFINE_REMOTE_LU

FULLY_QUALIFIED_LU_NAME specifies the fully qualified name of the remote LU. If the specified name is currently undefined to the local LU, this verb defines the remote LU's fully qualified name and initializes the other parameter values specified on this verb. If the specified name is already defined to the local LU, this verb changes the other parameter values.

LOCALLY_KNOWN_LU_NAME specifies the locally known name of the remote LU that local transaction programs can specify on the LU_NAME parameter of the MC_ALLOCATE and ALLOCATE verbs.

- **NONE** specifies that no locally known LU name is to be defined.
- **NAME** specifies the locally known LU name of the remote LU. This name is not sent outside the local LU.

UNINTERPRETED_LU_NAME specifies the uninterpreted LU name of the remote LU, which the local LU uses on INITIATE and TERMINATE requests it sends to its SSCP.

- **NONE** specifies that no uninterpreted LU name is to be defined.
- **NAME** specifies the uninterpreted LU name of the remote LU.

INITIATE_TYPE specifies the session-initiation type that the local LU is to use on INITIATE requests it sends to its SSCP for initiating sessions with the remote LU.

- **INITIATE_ONLY** specifies that session initiation requests are to indicate "initiate only." The SSCP will not queue the session initiation requests.
- **INITIATE_OR_QUEUE** specifies that session initiation requests are to indicate "initiate or queue." The SSCP may queue session the initiation requests, if necessary, while waiting for the remote LU to become available.

PARALLEL_SESSION_SUPPORT specifies whether the local LU supports parallel sessions with the remote LU. The local LU uses this parameter to determine the indication for parallel session support that it specifies in session activation (BIND) requests and responses.

- **YES** specifies that parallel sessions are supported.
- **NO** specifies that parallel sessions are not supported.

CNOS_SUPPORT specifies whether the local LU supports the exchange of CNOS requests and replies with the remote LU. The local LU uses this parameter to determine the indication for CNOS support that it specifies in session activation (BIND) requests and responses.

- **YES** specifies that CNOS is supported.
PARALLEL_SESSION_SUPPORT(YES) must also be specified.
- **NO** specifies that CNOS is not supported.
PARALLEL_SESSION_SUPPORT(NO) must also be specified.

LU_LU_PASSWORD specifies the LU-LU password to be used for session-level LU-LU verification during session activation. The LU-LU password must be the same as that defined at the remote LU.

- **NONE** specifies that no LU-LU password is to be defined.

DEFINE_REMOTE_LU

- **VALUE** specifies the LU-LU password. It must be a random binary value up to 64 bits (8 bytes) in length. It should be specified in a form that can yield any binary value. For example, it could be specified using the hexadecimal digits 0, 1, 2, ..., E, F to represent each group of 4 bits. After being defined, the LU-LU password is nondisplayable.

SECURITY_ACCEPTANCE specifies the level of access security information that the local LU will accept on allocation requests it receives from the remote LU. Access security information that includes a password is verified against the LU's conversation-level security verification list prior to acceptance, as described for the **SECURITY** parameter on the **DEFINE_LOCAL_LU** verb.

- **NONE** specifies that no access security information is to be accepted on allocation requests received from the remote LU.
- **CONVERSATION** specifies that the local LU will accept conversation-level access security information, which must include both a user ID and password, and may also include a profile. The local LU will not accept remotely initiated allocation requests that include the already-verified or persistent-verification indications.
- **ALREADY_VERIFIED** specifies that the local LU will accept conversation-level access security information, which may include the already-verified indication in place of a password. The local LU will not accept remotely initiated allocation requests that include a persistent-verification indication.
- **PERSISTENT_VERIFICATION** specifies that the local LU will accept conversation-level access security information, which may include a persistent-verification indication. The local LU will not accept remotely initiated allocation requests that include the already-verified indication.
- **AV_AND_PV** specifies that the local LU will accept conversation-level access security information, as well as the already-verified and persistent-verification indications on remotely initiated allocation requests.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- **OK**
- **PARAMETER_ERROR**, for one of the following reasons:
 - **FULLY_QUALIFIED_LU_NAME** specifies a value that is not a type-A symbol string.
 - **PARALLEL_SESSION_SUPPORT(YES)** is specified and the total LU-LU session limit is 1.
 - **PARALLEL_SESSION_SUPPORT(YES)** and **CNOS_SUPPORTED(NO)** are specified.
 - **PARALLEL_SESSION_SUPPORT(NO)** and **CNOS_SUPPORTED(YES)** are specified.
 - **PARALLEL_SESSION_SUPPORT**, **CNOS_SUPPORT**, **LU_LU_PASSWORD**, or **SECURITY_ACCEPTANCE** is specified and at least one (LU,mode) session limit for the remote LU is not 0, or the LU-LU session count between the local and remote LUs is not 0.

DEFINE_REMOTE_LU

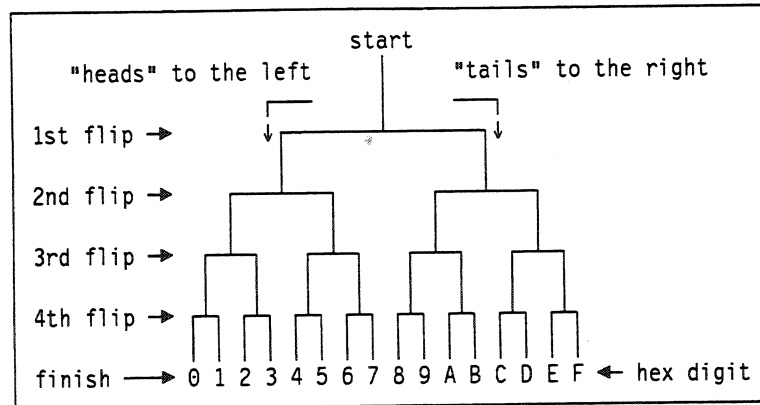
- PROGRAM_PARAMETER_CHECK, for the following reason:
 - The program issuing this verb does not have define privilege.

Notes:

1. This verb can be used to define the fully qualified name of a remote LU. In this case, the verb should be issued prior to the local LU participating in any network activity involving the remote LU.
2. If no locally known name of the remote LU is defined at the local LU, local transaction programs may specify the remote LU's fully qualified LU name, or its uninterpreted name if one is currently defined at the local LU, on the LU_NAME parameter of the MC_ALLOCATE and ALLOCATE verbs.
3. An uninterpreted name of the remote LU must be defined at both the local LU and its SSCP before the local LU sends INITIATE and TERMINATE requests to its SSCP.
4. If no initiate type is defined at the local LU for the remote LU and the product LU sends INITIATE requests to its SSCP, the type used on the requests is product-determined.
5. Parallel-session support must be defined at the local LU for the remote LU before it activates sessions with the remote LU.
6. CNOS support must be defined at the local LU for the remote LU before it activates sessions with the remote LU.
7. Session-level LU-LU verification is used to verify the identity of each LU to its session partner LU during activation of an LU-LU session. It uses an LU-LU password as the key to the Data Encryption Standard (DES) algorithm, in conjunction with an LU-generated random-data value carried on the session-activation request and response.

The same LU-LU password specification must be defined at both LUs, either NONE or an LU-LU password. An LU-LU password should be a random binary value. The means for specifying the LU-LU password is product-dependent. The product may provide a utility procedure for generating an LU-LU password, or it may require the user to enter the password manually. In the latter case, the human operator may use the following method to produce a random value:

- Enter the password value by means of hex digits (the numerals 0 through 9 and the upper-case characters A through F).
- Enter 16 random hex digits (fewer digits or non-random digits will reduce the effective security for LU-LU verification).
- For each of the 16 hex digits, flip a coin four times. At each flip of the coin, follow the path illustrated in the following figure. The fourth flip will select the hex digit to be used. Repeat this procedure until all 16 hex digits are obtained. The result is a 64-bit random LU-LU password.



The total number of coin flips is 64. Of course, the equivalent value for the LU-LU password can be obtained in binary notation, where each coin flip, of "heads" or "tails," selects the next binary digit, 0 or 1, respectively.

8. If no LU-LU password is defined at the local LU for the remote LU, no session-level LU-LU verification will take place between the two LUs.
9. Conversation-level access security information is carried on allocation requests in order for the receiving LU to verify the identity of the user ID, and to control access to its resources. The information includes a user ID together with a password, the already-verified indication, or a persistent-verification indication; the information may also include a profile. Allocation requests that include a password are verified against the LU's conversation-level security verification list; see the SECURITY parameter on the DEFINE_LOCAL_LU verb for more details about conversation-level security verification. The already-verified indication signifies that the identity of the user ID has already been verified.
10. If no conversation-level security is defined at the local LU for the remote LU, the local LU will accept from the remote LU only allocation requests that carry no access security information.
11. More details concerning the LU's use of these operating parameters are given in *SNA LU 6.2 Reference: Peer Protocols*.

DEFINE_TP

DEFINE_TP

Initializes or changes parameters that control the operation of the local LU in conjunction with a transaction program.

DEFINE_TP	Option No. [505]
Supplied Parameters:	
TP_NAME (variable)	
STATUS	
(ENABLED)	
(TEMP_DISABLED)	
(PERM_DISABLED)	
CONVERSATION_TYPE	
(MAPPED)	
(BASIC)	
(MAPPED_OR_BASIC)	
(FDX_MAPPED)	112
(FDX_BASIC)	112
(FDX_MAPPED_OR_BASIC)	112
(ANY)	112
SYNC_LEVEL	
(NONE)	
(CONFIRM)	
(SYNCPT)	108
SECURITY_REQUIRED	
(NONE)	
(CONVERSATION)	212
(ACCESS (USER_ID)	214
(PROFILE)	215
(USER_ID_PROFILE)	214, 215
(USER_ID_REMOTE_LU)	214, 216
(USER_ID_PROFILE_REMOTE_LU))	214, 215, 216
SECURITY_ACCESS	
(ADD (PROFILE (variable))	215
(REMOTE_LU (variable))	216
(USER_ID (variable)))	214
(DELETE (PROFILE (variable))	215
(REMOTE_LU (variable))	216
(USER_ID (variable)))	214
PIP (NO)	
(YES (variable))	241
(NO_LU_VERIFICATION)	

DATA_MAPPING (NO) (YES)	246
PRIVILEGE (NONE) (CNOS) (SESSION_CONTROL) (DEFINE) (DISPLAY) (ALLOCATE_SERVICE_TP)	502 or 504
INSTANCE_LIMIT (variable)	401

Returned Parameters:

RETURN_CODE (variable)

Supplied Parameters:

TP_NAME specifies the local transaction program name. If the specified name is not currently defined at the local LU, this verb defines the program name and initializes the other parameter values specified on this verb. If the specified name is already defined to the local LU, this verb changes the other parameter values.

STATUS specifies the status for starting execution of the transaction program when the local LU receives an allocation request naming the program.

- **ENABLED** specifies that the local LU can start the program.
- **TEMP_DISABLED** specifies that the local LU cannot start the program. The local LU rejects the allocation request with an indication that the program is not available but retry is possible.
- **PERM_DISABLED** specifies that the local LU cannot start the program. The local LU rejects the allocation request with an indication that the program is not available and no retry is possible.

CONVERSATION_TYPE specifies the conversation type allowed on allocation requests that start the transaction program.

- **MAPPED** specifies that allocation requests indicating mapped conversation are allowed to start the program.
- **BASIC** specifies that allocation requests indicating basic conversation are allowed to start the program.
- **MAPPED_OR_BASIC** specifies that allocation requests indicating either mapped conversation or basic conversation are allowed to start the program.
- **FDX_MAPPED** specifies that allocation requests indicating full-duplex mapped conversation are allowed to start the program.

DEFINE_TP

- **FDX_BASIC** specifies that allocation requests indicating full-duplex basic conversation are allowed to start the program.
- **FDX_MAPPED_OR_BASIC** specifies that allocation requests indicating either full-duplex mapped conversation or full-duplex basic conversation are allowed to start the program.
- **ANY** specifies that allocation requests indicating any of the above conversation types are allowed to start the program.

SYNC_LEVEL specifies the synchronization level allowed on allocation requests that start the transaction program.

- **NONE** specifies that allocation requests indicating a synchronization level of none are allowed to start the program.
- **CONFIRM** specifies that allocation requests indicating a synchronization level of confirm are allowed to start the program.
- **SYNCPT** specifies that allocation requests indicating a synchronization level of sync point are allowed to start the program.

Any combination of these arguments may be specified.

SECURITY_REQUIRED specifies the type of security verification required to be performed on incoming allocation requests that designate the transaction program. (Conversation-level security verification, when required, is performed as specified on the **SECURITY** parameter of the **DEFINE_LOCAL_LU** verb.)

- **NONE** specifies that no verification is required. Allocation requests designating the transaction program may omit or include access security information. Conversation-level security verification will be performed on those requests that include a user ID and password, but no resource-access verification is performed.
- **CONVERSATION** specifies that conversation-level security verification is to be performed on requests that carry a user ID and password, but no resource-access verification is performed. Allocation requests designating the transaction program must carry a user ID and either a password or an already-verified indication. (Acceptance of the already-verified indication is determined by the **SECURITY_ACCEPTANCE** parameter of the **DEFINE_REMOTE_LU** verb.)
- **ACCESS** specifies that conversation-level security verification is to be performed on requests that carry a user ID and password, and resource-access verification is also to be performed. Allocation requests designating the transaction program must carry a user ID and either a password or an already-verified indication. (Acceptance of the already-verified indication is determined by the **SECURITY_ACCEPTANCE** parameter of the **DEFINE_REMOTE_LU** verb.) The local LU performs resource-access verification using a resource-access authorization list associated with the transaction program. The list is created by means of the **SECURITY_ACCESS** parameter. The type of resource-access verification to be performed is specified as follows:
 - **PROFILE** specifies that the profile carried on the allocation request is to be verified against the resource-access authorization list. The allocation request must carry a profile that matches one in the authorization list. The user ID

on the allocation request and the identity of the remote LU that sent the request are ignored for the resource-access verification.

- **USER_ID** specifies that the user ID carried on the allocation request is to be verified against the resource-access authorization list. The allocation request must carry a user ID that matches one in the authorization list. The profile (if present) on the allocation request and the identity of the remote LU that sent the request are ignored for the resource-access verification.
- **USER_ID_PROFILE** specifies that the user ID and profile carried on the allocation request are to be verified against the resource-access authorization list. The allocation request must carry a user ID and profile that match a user ID and associated profile in the authorization list. The identity of the remote LU that sent the request is ignored for the resource-access verification.
- **USER_ID_REMOTE_LU** specifies that the user ID carried on the allocation request and the identity of the remote LU that sent the request are to be verified against the resource-access authorization list. The allocation request must carry a user ID that matches one in the authorization list, and the name of the remote LU that sent the request must match a remote LU name associated with the user ID in the authorization list. The profile (if present) on the allocation request is ignored for the resource-access verification.
- **USER_ID_PROFILE_REMOTE_LU** specifies that the user ID and profile carried on the allocation request and the identity of the remote LU that sent the request are to be verified against the resource-access authorization list. The allocation request must carry a user ID and profile that match a user ID and associated profile in the authorization list, and the name of the remote LU that sent the request must match a remote LU name associated with the user ID in the authorization list.

SECURITY_ACCESS specifies to add or delete access security information that the local LU uses for resource-access verification. This parameter must be specified when the **SECURITY_REQUIRED** parameter specifies that resource-access verification is required. The local LU updates a resource-access authorization list, associated with the transaction program, from the information supplied on this parameter. The resource-access authorization list consists of either (1) one or more profiles, or (2) one or more user IDs with zero or more profiles and remote LU names associated with each user ID.

- **ADD** specifies to add access security information to the resource-access authorization list associated with the transaction program. A profile may be specified alone, or a user ID may be specified alone or together with a profile, a remote LU name, or both.
- **PROFILE** specifies a profile to be added to the resource-access authorization list. A profile may be specified only when the **SECURITY_REQUIRED** parameter specifies resource-access verification that includes verification of profiles. A profile must be specified alone when the resource-access verification includes verification of only profiles. A user ID must also be specified when the resource-access verification includes ver-

DEFINE_TP

ification of both user IDs and profiles. If a user ID is also specified, the profile is added to those associated with the user ID.

- **REMOTE_LU** specifies the fully qualified name of a remote LU to be added to the resource-access authorization list. A remote LU name may be specified only when the **SECURITY_REQUIRED** parameter specifies resource-access verification that includes verification of the identity of remote LUs. A user ID must also be specified. The remote LU name is added to those associated with the user ID.
- **USER_ID** specifies a user ID. A user ID must be specified when the **SECURITY_REQUIRED** parameter specifies resource-access verification that includes verification of user IDs. A profile or remote LU name, or both, may also be specified, depending on the **SECURITY_REQUIRED** parameter. If the user ID is not currently defined for the transaction program, the LU adds it and the profile and remote LU name (if specified) to the resource-access authorization list. If the user ID is already defined, the list is updated with the profile or remote LU name (or both if both are specified).
- **DELETE** specifies to delete access security information from the resource-access authorization list associated with the transaction program. A profile may be specified alone, or a user ID may be specified alone or together with a profile, a remote LU name, or both.
 - **PROFILE** specifies a profile to be deleted from the resource-access authorization list. A profile may be specified only when the **SECURITY_REQUIRED** parameter specifies resource-access verification that includes verification of profiles. A profile must be specified alone when the resource-access verification includes verification of only profiles. A user ID must also be specified when the **SECURITY_REQUIRED** parameter specifies resource-access verification that includes verification of both user IDs and profiles.
 - **REMOTE_LU** specifies the fully qualified name of a remote LU to be deleted from the resource-access authorization list. A remote LU name may be specified only when the **SECURITY_REQUIRED** parameter specifies resource-access verification that includes verification of the identity of remote LUs. A user ID must also be specified.
 - **USER_ID** specifies a user ID. A user ID must be specified when the **SECURITY_REQUIRED** parameter specifies resource-access verification that includes verification of user IDs. A profile or remote LU name, or both, may also be specified, depending on the **SECURITY_REQUIRED** parameter. If a user ID is specified alone, the user ID and all of its associated profiles and remote LU names are deleted from the resource-access authorization list. If a profile or remote LU name is also specified, the user ID and profile or remote LU name (or all three if they are all specified) are deleted when no other profiles and remote LU names are currently defined for the user ID; if other profiles or remote LU names are currently defined for the user ID, the user ID and other profiles and remote LU names remain defined.

PIP specifies whether PIP data is required on allocation requests that start the transaction program.

- **NO** specifies that no PIP data is required. Only allocation requests carrying no PIP data are allowed to start the program.
- **YES** specifies that PIP data is required. Only allocation requests carrying the number of PIP subfields specified on this parameter are allowed to start the program. The specified number should agree with the number of PIP variables associated with the program. For more information about the association of PIP variables with the program, see "Transaction Program Structure and Execution" on page 1-8
- **NO_LU_VERIFICATION** specifies that the LU is to perform no verification of the number of PIP subfields received versus the number expected. The transaction program is responsible for verifying the validity of the received PIP data. Because a transaction program could be invoked for multiple reasons, it is possible that on one invocation it is to be passed some number of program initialization parameters (PIP data), and on another invocation it is passed a different number (possibly zero) because it is being invoked for a different reason.

DATA_MAPPING specifies whether data mapping support is to be provided to the transaction program. This parameter applies only when **CONVERSATION_TYPE(MAPPED)** or **CONVERSATION_TYPE(MAPPED|BASIC)** is also specified.

- **NO** specifies that no data mapping support is to be provided. Map names received on any mapped conversations allocated to the program are rejected.
- **YES** specifies that data mapping support is to be provided.

PRIVILEGE specifies the category of control operator verbs that the transaction program is allowed to issue. Either **NONE** or any combination of **CNOS**, **SESSION_CONTROL**, **DEFINE**, **DISPLAY**, and **ALLOCATE_SERVICE_TP** may be specified.

- **NONE** specifies that the program is not allowed to issue verbs that require a privilege to do so.
- **CNOS** specifies that the program is allowed to issue the **CNOS** verbs.
- **SESSION_CONTROL** specifies that the program is allowed to issue the **ACTIVATE_SESSION** and **DEACTIVATE_SESSION** verbs.
- **DEFINE** specifies that the program is allowed to issue the **DEFINE** verbs and the **DELETE** verbs.
- **DISPLAY** specifies that the program is allowed to issue the **DISPLAY** verbs.
- **ALLOCATE_SERVICE_TP** specifies that the program is allowed to issue the **ALLOCATE** verb with its **TPN** parameter specifying an SNA service transaction program.

INSTANCE_LIMIT specifies the maximum number of instances of the transaction program that are allowed to be invoked concurrently.

Returned Parameters:

DEFINE_TP

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- OK
- **PARAMETER_ERROR**, for one of the following reasons:
 - **CONVERSATION_TYPE**(BASIC) or (**FULL_DUPLEX_BASIC**) and either **DATA_MAPPING**(YES) or **USER_CONTROL_DATA**(YES) are specified.
 - **SECURITY_ACCESS** is specified, **SECURITY_REQUIRED** is omitted, and no type of security verification is currently defined.
 - **SECURITY_ACCESS** is specified and the type of security verification specified on **SECURITY_REQUIRED** or currently defined does not include resource-access verification.
 - **SECURITY_ACCESS** specifies a user ID, profile, or remote LU name and the type of security verification specified on **SECURITY_REQUIRED** or currently defined does not include the respective verification of user IDs, profiles, or remote LU names.
 - **SECURITY_ACCESS** specifies only a profile and the type of security verification specified on **SECURITY_REQUIRED** or currently defined includes verification of user IDs.
 - **SECURITY_ACCESS** specifies only a remote LU name.
 - **SECURITY_ACCESS**(ADD(...)) specifies a user ID or profile that is not a symbol-string type (A, AE, GR, or DB) that the product supports.
 - **SECURITY_ACCESS**(ADD(...)) specifies a remote LU name that is not a type-A symbol string.
 - **SECURITY_ACCESS**(DELETE(...)) specifies a user ID, profile, or remote LU name that is not currently defined at the local LU.
 - **SYNC_LEVEL**(CONFIRM) is not supported if only full-duplex conversations are specified.
- **PROGRAM_PARAMETER_CHECK**, for the following reason:

The program issuing this verb does not have define privilege.

Notes:

1. The values specified on the parameters of this verb take effect at the next invocation of the transaction program.
2. If the conversation type allowed on allocation requests that start the transaction program is not defined, a mapped or basic conversation is allowed for either full-duplex or half-duplex.
3. If the synchronization level allowed on allocation requests that start the transaction program is not defined, a synchronization level of NONE or CONFIRM is allowed. A synchronization level of CONFIRM is not supported for full-duplex conversations.
4. Resource-access verification is used to verify the access security information on incoming allocation requests for the authority to access the transaction program named on the requests and the local resources that the program allocates. The local LU maintains a resource-access authorization list for this purpose. The authorization list is created and updated from information supplied on the **SECURITY_ACCESS** parameter. The list may consist of profiles alone, or it

may consist of user IDs alone or with associated profiles or remote LU names or both, as determined by the SECURITY_REQUIRED parameter.

5. There is a resource-access authorization list for each transaction program for which resource-access verification is defined. When the first user ID and associated password and profiles are added, or the first profile is added, a resource-access authorization list is created for the program. When the last user ID and associated password and profiles are deleted, or the last profile is deleted, the resource-access authorization list itself is deleted.
6. If resource-access verification is to be performed and it includes verification of user IDs, the authorization list must contain one or more user IDs when the verification of allocation requests takes place. Similarly, if the verification includes profiles, the list must contain one or more profiles; and if the verification includes remote LU names, the list must contain one or more remote LU names.
7. If the type of security verification required on incoming allocation requests is currently defined and a different type is specified, the new type replaces the current type. If resource-access verification is currently defined as being required and a different type of resource-access verification is specified, the resource-access authorization list is deleted and a new list is created.
8. If SECURITY_ACCESS is specified without SECURITY_REQUIRED, the type of security verification currently defined applies to the use of the SECURITY_ACCESS parameter.
9. If no type of security verification is currently defined, none is required to start the program. However, if the allocation request carries access security information, the local LU performs conversation-level security verification.
10. If no privilege is defined at the local LU for the transaction program, the program may issue only conversation verbs.
11. More details concerning the LU's use of these operating parameters are given in *SNA LU 6.2 Reference: Peer Protocols*.

DELETE

DELETE

Deletes parameter values, established by means of the DEFINE verbs, that control the operation of the local LU. The execution of this verb involves only the local LU; it does not cause any information to be sent outside the LU.

DELETE	Option No. [505]
Supplied Parameters:	
One of the below parameters is to be selected to identify the deletion.	
LOCAL_LU_NAME (variable) REMOTE_LU_NAME (variable) MODE_NAME (variable) TP_NAME (variable)	
Returned Parameters:	
RETURN_CODE (variable)	

Supplied Parameters:

LOCAL_LU_NAME specifies the fully qualified name of the local LU. This parameter must be specified alone. Specifying this parameter deletes the local LU name and all parameter values associated with the local LU; that is, it deletes all parameter values that have been defined by means of the DEFINE_LOCAL_LU, DEFINE_REMOTE_LU, DEFINE_MODE, and DEFINE_TP verbs.

REMOTE_LU_NAME specifies the fully qualified name of the remote LU. This parameter may be specified together with the MODE_NAME parameter. Specifying this parameter without the MODE_NAME parameter deletes the remote LU name and all parameter values associated with the remote LU; that is, it deletes all parameter values that have been defined by means of the DEFINE_REMOTE_LU and DEFINE_MODE verbs. Specifying this parameter together with the MODE_NAME parameter deletes parameter values associated with the mode name, but the remote LU name and all parameter values not associated with the mode name remain unchanged.

MODE_NAME specifies the mode name. This parameter must be specified together with the REMOTE_LU_NAME parameter. Specifying this parameter deletes all parameter values associated with the mode name for the remote LU; that is, it deletes the mode name and all parameter values that have been defined by means of the DEFINE_MODE_NAME verb.

TP_NAME specifies the local transaction program name. Specifying this parameter deletes all parameter values associated with the transaction program; that is, it deletes the program name and all parameter values that have been defined by means of the **DEFINE_TP** verb.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- OK
- **PARAMETER_ERROR**, for one of the following reasons:
 - **LOCAL_LU_NAME** specifies a local LU name not currently defined at the local LU.
 - **LOCAL_LU_NAME** is not specified alone.
 - **REMOTE_LU_NAME** specifies a remote LU name not currently defined at the local LU.
 - **MODE_NAME** specifies a mode name not currently defined at the local LU.
 - **MODE_NAME** is specified without **REMOTE_LU_NAME**.
 - **TP_NAME** specifies a local transaction program name not currently defined at the local LU.
 - The item to be deleted is active at this time and cannot be deleted.
- **PROGRAM_PARAMETER_CHECK**, for the following reason:
 - The program issuing this verb does not have define privilege.

Notes:

1. Deleting parameter values makes those values undefined to the local LU.
2. When deleting a local LU name and all its associated parameter values, verb should be issued only when the local LU is not participating in any network activity.
3. When deleting a remote LU name and all its associated parameter values, verb should be issued only when the local LU is not participating in any network activity involving the remote LU.
4. When deleting a mode name and all its associated parameter values, verb should be issued only when the local LU is not participating in any network activity involving the remote LU and mode name.
5. When deleting a transaction program name and all its associated parameter values, verb should be issued only when the transaction program is not in use.
6. More details concerning the LU's use of the LU-LU session limit are given in *SNA LU 6.2 Reference: Peer Protocols*.

DISPLAY_LOCAL_LU

DISPLAY_LOCAL_LU

Returns current values of parameters that control the operation of the local LU.

DISPLAY_LOCAL_LU	Option No. [505]
Supplied Parameters:	
FULLY_QUALIFIED_LU_NAME (variable)	
Returned Parameters:	
RETURN_CODE (variable)	
LU_SESSION_LIMIT (variable)	
LU_SESSION_COUNT (variable)	
SECURITY (variable)	212
MAP_NAMES (variable)	246
REMOTE_LU_NAMES (variable)	
TP_NAMES (variable)	
BIND_RSP_QUEUE_CAPABILITY (variable)	
LENGTH_CHECKED_COMPRESSION (variable)	618
INBOUND_COMPRESSION_LEVEL (variable)	618
OUTBOUND_COMPRESSION_LEVEL (variable)	618
RLE_BEFORE_LZ (variable)	618

Supplied Parameters:

FULLY_QUALIFIED_LU_NAME specifies the fully (or *network-*) qualified name of the local LU.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- OK
- **PARAMETER_ERROR**, for the following reason:
 - **FULLY_QUALIFIED_LU_NAME** does not specify a local LU name currently defined at the local LU.
- **PROGRAM_PARAMETER_CHECK**, for the following reason:
 - The program issuing this verb does not have display privilege.

LU_SESSION_LIMIT returns the LU-LU session limit currently defined at the local LU.

LU_SESSION_COUNT returns the LU-LU session count, which is the total number of active sessions for the local LU.

SECURITY returns the conversation-level security verification list currently defined at the local LU.

MAP_NAMES returns a list of the local map names currently defined at the local LU.

REMOTE_LU_NAMES returns a list of the remote LU names currently defined at the local LU.

TP_NAMES returns a list of the local transaction program names currently defined at the local LU.

BIND_RSP_QUEUE_CAPABILITY returns an indication of whether the local LU is willing to allow the remote LU to queue session activation (BIND) requests if a session cannot be activated immediately.

LENGTH_CHECKED_COMPRESSION returns an indication of the compression support on sessions involving this LU.

- **PROHIBITED** specifies that no compression is to be used.
- **REQUIRED** specifies that compression is required on each session.
- **NEGOTIABLE** specifies that the usage of compression is to be negotiated by the PLU and the SLU. If compression is to be used, the level of compression will be negotiated.

INBOUND_COMPRESSION_LEVEL returns the maximum level of compression support for inbound data on sessions involving the local LU. If compression is prohibited for the local LU, this parameter will return *NONE*. Other possible values are *RLE*, *LZ9*, *LZ10*, and *LZ12*. (See "DEFINE_LOCAL_LU" on page 10-3 for an explanation of the values.)

OUTBOUND_COMPRESSION_LEVEL returns the maximum level of compression support for outbound data on sessions involving the local LU. If compression is prohibited for the local LU, this parameter will return *NONE*. Other possible values are *RLE*, *LZ9*, *LZ10*, and *LZ12*.

RLE_BEFORE_LZ returns an indication of *YES* if RLE before LZ compression on sessions involving the local LU is supported; *NO* otherwise. Support for this parameter is implementation-dependent.

Notes:

1. This verb can be used to obtain operating parameter values that are established by the **DEFINE_LOCAL_LU**, **DEFINE_REMOTE_LU**, and **DEFINE_TP** verbs, as well as the current LU-LU session count.
2. More details concerning the LU's use of these operating parameters are given in *SNA LU 6.2 Reference: Peer Protocols*.
3. See Note 6 on page 8-6 regarding intra-LU sessions.
4. The values returned by the length-checked compression parameters

DISPLAY_LOCAL_LU

LENGTH_CHECKED_COMPRESSION
INBOUND_COMPRESSION_LEVEL
OUTBOUND_COMPRESSION_LEVEL
RLE_BEFORE_LZ

are those given on the most recent DEFINE_LOCAL_LU to set these parameters. Also, these values apply only to sessions for which the local LU is the PLU (see "DEFINE_LOCAL_LU" on page 10-3).

DISPLAY_MODE

Returns current values of parameters that control the operation of the local LU in conjunction with a group of sessions to a remote LU, the session group being identified by a mode name.

DISPLAY_MODE	Option No. [505]
Supplied Parameters:	
FULLY_QUALIFIED_LU_NAME (variable)	
MODE_NAME (variable)	
Returned Parameters:	
RETURN_CODE (variable)	
SEND_MAX_RU_SIZE_LOWER_BOUND (variable)	
610	
SEND_MAX_RU_SIZE_UPPER_BOUND (variable)	
610	
RECEIVE_MAX_RU_SIZE_LOWER_BOUND (variable)	
610	
RECEIVE_MAX_RU_SIZE_UPPER_BOUND (variable)	
610	
SINGLE_SESSION_REINITIATION (variable)	
608	
SESSION_LEVEL_CRYPTOGRAPHY (variable)	
611, 617	
LENGTH_CHECKED_COMPRESSION (variable)	
618	
INBOUND_COMPRESSION_LEVEL (variable)	
618	
OUTBOUND_COMPRESSION_LEVEL (variable)	
618	
RLE_BEFORE_LZ (variable)	
618	
CONWINNER_AUTO_ACTIVATE_LIMIT (variable)	
612	
LOCAL_MAX_SESSION_LIMIT (variable)	
613	
LU_MODE_SESSION_LIMIT (variable)	
MIN_CONWINNERS (variable)	
MIN_CONLOSERS (variable)	
TERMINATION_COUNT (variable)	
DRAIN_LOCAL_LU (variable)	
DRAIN_REMOTE_LU (variable)	
LU_MODE_SESSION_COUNT (variable)	
CONWINNERS_SESSION_COUNT (variable)	
CONLOSERS_SESSION_COUNT (variable)	
SESSION_IDS (variable)	
CONVERSATION_GROUP_IDS (variable)	
401	
PREFERRED_RECEIVE_RU_SIZE (variable)	
610	
PREFERRED_SEND_RU_SIZE (variable)	
610	
SESSION_DEACTIVATED_TP_NAME (variable)	
401	

DISPLAY_MODE

Supplied Parameters:

FULLY_QUALIFIED_LU_NAME specifies the fully (or *network-*) qualified name of the remote LU.

MODE_NAME specifies the mode name.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- OK
- **PARAMETER_ERROR**, for one of the following reasons:
 - **FULLY_QUALIFIED_LU_NAME** does not specify a remote LU name currently defined at the local LU.
 - **MODE_NAME** does not specify a mode name currently defined at the local LU.
- **PROGRAM_PARAMETER_CHECK**, for the following reason:
 - The program issuing this verb does not have display privilege.

SEND_MAX_RU_SIZE_LOWER_BOUND returns the lower bound for the maximum size of normal-flow requests that the local LU sends on the sessions, currently defined at the local LU.

SEND_MAX_RU_SIZE_UPPER_BOUND returns the upper bound for the maximum size of normal-flow requests that the local LU sends on the sessions, currently defined at the local LU.

RECEIVE_MAX_RU_SIZE_LOWER_BOUND returns the lower bound for the maximum size of normal-flow requests that the local LU receives on the sessions, currently defined at the local LU.

RECEIVE_MAX_RU_SIZE_UPPER_BOUND returns the upper bound for the maximum size of normal-flow requests that the local LU receives on the sessions, currently defined at the local LU.

SINGLE_SESSION_REINITIATION returns an indication of the session reinitiation responsibility for a single session with the remote LU. If a session is active between the local and remote LUs, this parameter returns an indication of the actual session reinitiation responsibility; otherwise, it returns an indication of the responsibility currently defined at the local LU.

SESSION_LEVEL_CRYPTOGRAPHY returns an indication of the session-level cryptography support for the sessions. If one or more sessions within the mode name group are active between the local and remote LUs, this parameter returns an indication of the actual session-level cryptography support; otherwise, it returns an indication of the support currently defined at the local LU.

LENGTH_CHECKED_COMPRESSION returns an indication of the compression support on sessions using this mode.

- **PROHIBITED** specifies that no compression is to be used.

- **REQUIRED** specifies that compression is required on each session using this mode.
- **NEGOTIABLE** specifies that the usage of compression is to be negotiated by the PLU and the SLU. If compression is to be used, the level of compression will be negotiated.

INBOUND_COMPRESSION_LEVEL returns the maximum level of compression support for inbound data on sessions using this mode. If compression is prohibited with this mode, this parameter will return *NONE*. Other possible values are *RLE*, *LZ9*, *LZ10*, and *LZ12*. (See "DEFINE_MODE" on page 10-9 for an explanation of the values.)

OUTBOUND_COMPRESSION_LEVEL returns the maximum level of compression support for outbound data on sessions using this mode. If compression is prohibited with this mode, this parameter will return *NONE*. Other possible values are *RLE*, *LZ9*, *LZ10*, and *LZ12*.

RLE_BEFORE_LZ returns an indication of *YES* if RLE before LZ compression on sessions using this mode is supported; *NO*, otherwise. Support for this parameter is implementation-dependent.

CONWINNER_AUTO_ACTIVATE_LIMIT returns the local LU's automatic-activation limit on the number of contention winner sessions, currently defined at the local LU.

LOCAL_MAX_SESSION_LIMIT returns the current LU session limit, including all modes, for the local LU.

LU_MODE_SESSION_LIMIT returns the current (LU,mode) session limit.

MIN_CONWINNERS returns the current minimum number of sessions for which the local LU is designated to be the contention winner.

MIN_CONLOSERS returns the current minimum number of sessions for which the remote LU is designated to be the contention winner, making the local LU the contention loser.

TERMINATION_COUNT returns the termination count, which is the number of sessions for which that the local LU is responsible to deactivate as a result of CNOS processing.

DRAIN_LOCAL_LU returns an indication of whether the local LU is allowed to drain its allocation requests as a result of CNOS processing that resets the (LU,mode) session limit.

DRAIN_REMOTE_LU returns an indication of whether the remote LU is allowed to drain its allocation requests as a result of CNOS processing that resets the (LU,mode) session limit.

LU_MODE_SESSION_COUNT returns the current (LU,mode) session count.

CONWINNERS_SESSION_COUNT returns the number of active sessions for which the local LU is the contention winner.

CONLOSERS_SESSION_COUNT returns the number of active sessions for which the local LU is the contention loser.

SESSION_IDS returns a list of the session identifiers assigned to the active sessions.

DISPLAY_MODE

CONVERSATION_GROUP_IDS returns a list of the conversation group IDs associated with the active sessions.

PREFERRED_RECEIVE_RU_SIZE returns the preferred value for the maximum size of normal-flow requests that the local LU receives on the sessions, currently defined at the local LU.

PREFERRED_SEND_RU_SIZE returns the preferred value for the maximum size of normal-flow requests that the local LU sends on the sessions, currently defined at the local LU.

SESSION_DEACTIVATED_TP_NAME returns the name of the transaction program to be invoked when a session is deactivated.

Notes:

1. This verb can be used to obtain operating parameter values that are established by the **DEFINE_MODE** verb and the **CNOS** verbs.
2. More details concerning the LU's use of these operating parameters are given in *SNA LU 6.2 Reference: Peer Protocols*.
3. See Note 6 on page 8-6 regarding intra-LU sessions.

DISPLAY_REMOTE_LU

Returns current values of parameters that control the operation of the local LU in conjunction with a remote LU.

DISPLAY_REMOTE_LU	Option No. [505]
Supplied Parameters:	
FULLY_QUALIFIED_LU_NAME (variable)	
Returned Parameters:	
RETURN_CODE (variable)	
LOCALLY_KNOWN_LU_NAME (variable)	606
UNINTERPRETED_LU_NAME (variable)	607
INITIATE_TYPE (variable)	
PARALLEL_SESSION_SUPPORT (variable)	
CNOS_SUPPORT (variable)	
SECURITY_ACCEPTANCE_LOCAL_LU (variable)	212
SECURITY_ACCEPTANCE_REMOTE_LU (variable)	212
MODE_NAMES (variable)	

Supplied Parameters:

FULLY_QUALIFIED_LU_NAME specifies the fully qualified name of the remote LU.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- OK
- PARAMETER_ERROR, for the following reason:
 - FULLY_QUALIFIED_LU_NAME does not specify a remote LU name currently defined at the local LU.
- PROGRAM_PARAMETER_CHECK, for the following reason:
 - The program issuing this verb does not have display privilege.

LOCALLY_KNOWN_LU_NAME returns the locally-known name of the remote LU, currently defined at the local LU.

DISPLAY_REMOTE_LU

UNINTERPRETED_LU_NAME returns the uninterpreted name of the remote LU, currently defined at the local LU.

INITIATE_TYPE returns an indication of the session-initiation type for the remote LU, currently defined at the local LU.

PARALLEL_SESSION_SUPPORT returns an indication of the parallel session support for sessions with the remote LU. If one or more sessions are active between the local and remote LUs, this parameter returns an indication of the actual parallel session support; otherwise, it returns an indication of the support currently defined at the local LU.

CNOS_SUPPORT returns an indication of the CNOS support for sessions with the remote LU. If one or more sessions are active between the local and remote LUs, this parameter returns an indication of the actual CNOS support; otherwise, it returns an indication of the support currently defined at the local LU.

SECURITY_ACCEPTANCE_LOCAL_LU returns an indication of the level of access security information that the local LU will accept on allocation requests it receives from the remote LU, currently defined at the local LU.

SECURITY_ACCEPTANCE_REMOTE_LU returns an indication of the level of access security information that the remote LU will accept on allocation requests it receives from the local LU, when one or more sessions are active between the local and remote LUs. The value returned is what is currently defined at the remote LU, which is conveyed to the local LU during session activation.

MODE_NAMES returns a list of the mode names currently defined at the local LU for sessions with the remote LU.

Notes:

1. This verb is used to obtain operating parameter values that are defined by the **DEFINE_REMOTE_LU** and **DEFINE_MODE** verbs.
2. More details concerning the LU's use of these operating parameters are given in *SNA LU 6.2 Reference: Peer Protocols*.

DISPLAY_TP

Returns current values of parameters that control the operation of the local LU in conjunction with a transaction program.

DISPLAY_TP	<i>Option No.</i> [505]
Supplied Parameters:	
TP_NAME (<i>variable</i>)	
Returned Parameters:	
RETURN_CODE (<i>variable</i>) STATUS (<i>variable</i>) CONVERSATION_TYPE (<i>variable</i>) SYNC_LEVEL (<i>variable</i>) SECURITY_REQUIRED (<i>variable</i>) SECURITY_ACCESS (<i>variable</i>) PIP (<i>variable</i>) DATA_MAPPING (<i>variable</i>) PRIVILEGE (<i>variable</i>) INSTANCE_LIMIT (<i>variable</i>) INSTANCE_COUNT (<i>variable</i>) 108 212,214,215,216 212,214,215,216 241 246 401 401	

Supplied Parameters:

TP_NAME specifies the local transaction program name.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- OK
- **PARAMETER_ERROR**, for the following reason:
 - **TP_NAME** does not specify a transaction program name that is currently defined at the local LU.
- **PROGRAM_PARAMETER_CHECK**, for the following reason:
 - The program issuing this verb does not have display privilege.

STATUS returns an indication of the status for starting execution of the transaction program, as currently defined at the local LU.

DISPLAY_TP

CONVERSATION_TYPE returns an indication of the conversation type required on allocation requests that start the transaction program, as currently defined at the local LU.

Note: See the MC_ALLOCATE and ALLOCATE verb descriptions for an explanation of conversation types.

SYNC_LEVEL returns an indication of the synchronization level required on allocation requests that start the transaction program, as currently defined at the local LU.

Note: See the MC_ALLOCATE and ALLOCATE verb descriptions for an explanation of synchronization levels.

SECURITY_REQUIRED returns an indication of the type of security verification that is required to be performed on incoming allocation requests designating the transaction program.

SECURITY_ACCESS returns the resource-access authorization list currently defined for the transaction program at the local LU.

PIP returns the number of PIP subfields required on allocation requests that start the transaction program, as currently defined at the local LU. If the transaction program is defined as performing its own PIP subfields verification, an indication to that effect is returned.

DATA_MAPPING returns an indication of whether data mapping support is provided to the transaction program, as currently defined at the local LU.

PRIVILEGE returns an indication of the class of privileged verbs that the transaction program is allowed to issue, as currently defined at the local LU.

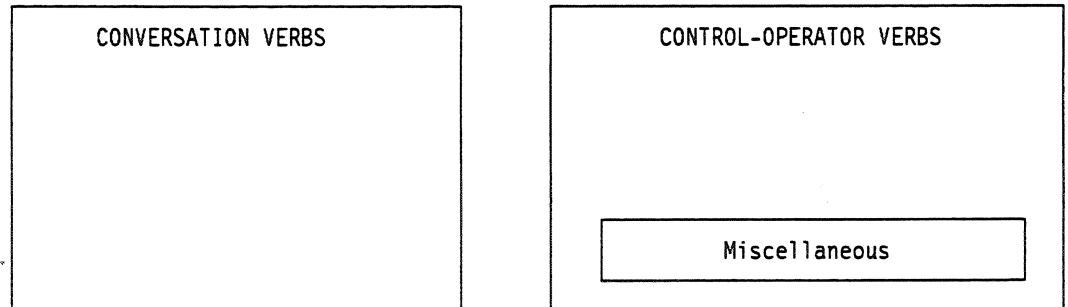
INSTANCE_LIMIT returns the number of instances of the transaction program allowed to be invoked concurrently, as currently defined at the local LU.

INSTANCE_COUNT returns the number of instances of the transaction program currently running at the local LU.

Notes:

1. This verb can be used to obtain operating parameter values that are established by the DEFINE_TP verb.
2. More details concerning the LU's use of these operating parameters are given in *SNA LU 6.2 Reference: Peer Protocols*.

Chapter 11. Miscellaneous COPS Verbs



Miscellaneous Verb Functions

This chapter describes any remaining control-operator verbs that don't fit into the CNOS, session control, or LU definition subcategories. The only function currently contained in this section is:

Security - Persistent Verification

Persistent Verification

This chapter describes a subcategory of miscellaneous control operator verbs consisting of security verbs. Persistent verification allows a user to verify its security information to the partner (one time) and then initiate consecutive conversations to the partner from that user ID. The following is the list of security verbs required for persistent verification:

Security - Persistent Verification

DISPLAY_SIGNED_ON_LIST
PROCESS_SIGNOFF
SIGNOFF

Only a transaction program that has privilege can issue the SIGNOFF verb. The SIGNOFF verb is used by the control operator to maintain the signed-on list.

Only a transaction program that has display privilege can issue the DISPLAY_SIGNED_ON_LIST verb. The program is designated to have display privilege when it is defined to the local LU (see the DEFINE_TP verb).

The execution of the DISPLAY_SIGNED_ON_LIST verb involves only the local LU, the execution of the SIGNOFF verb at the local LU causes the PROCESS_SIGNOFF to be processed at the target LU.

Miscellaneous verbs

The detailed descriptions of the persistent verification verbs follow, alphabetically by verb name.

DISPLAY_SIGNED_ON_LIST

Allows a program to retrieve the information the LU maintains in the signed-on lists. These lists are maintained to identify users signed on to the remote LUs from the local LU and those signed on to the local LU from remote LUs. Use of this verb requires display privilege as granted in the DEFINE_TP verb.

DISPLAY_SIGNED_ON_LIST	<i>Option No.</i> [219,220]
Supplied Parameters: LIST (<i>ALL_SIGNED_ON_LISTS</i>) (<i>SIGNED_ON_TO_LIST</i>) (<i>SIGNED_ON_FROM_LIST</i>)	
Returned Parameters: RETURN_CODE (<i>variable</i>) SIGNED_ON_TO_LIST (<i>variable</i>) ENTRIES_ON_TO_LIST (<i>variable</i>) SIGNED_ON_FROM_LIST (<i>variable</i>) ENTRIES_ON_FROM_LIST (<i>variable</i>)	

Supplied Parameters:

LIST specifies the name of the signed-on list from which information is desired.

- **SIGNED_ON_TO_LIST** specifies the list of users who are signed on to remote LUs from the local LU.
- **SIGNED_ON_FROM_LIST** specifies the list of remote users who are signed on to the local LU.
- **ALL_SIGNED_ON_LIST** specifies both the **SIGNED_ON_TO_LIST** and **SIGNED_ON_FROM_LIST**.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- OK
- **PROGRAM_PARAMETER_CHECK**, for the following reason:
 - The program issuing this verb does not have display privilege.

DISPLAY_SIGNED_ON_LIST

SIGNED_ON_TO_LIST specifies the variable into which the address of the list is placed.

ENTRIES_ON_TO_LIST specifies the variable in which is returned the number of elements, or entries, composing the requested list.

SIGNED_ON_FROM_LIST specifies the variable into which the address of the list is placed.

ENTRIES_ON_FROM_LIST specifies the variable in which is returned the number of elements, or entries, composing the requested list.

Notes:

1. The list returned is made up of list elements. Each element contains information about one entry in a signed-on list. Each element is composed of:
 - The user ID and profile (possibly a null string) from the sign-on Attach.
 - The local LU name of the partner LU from which or to which this user signed on.
 - The number of active conversations for this list element.
 - The finish time of the most recently completed conversation for this list element. A time of all 0s indicates no conversation started since sign-on has yet completed.
2. Information on conversations started before the most recent sign-on are not included, since sign-on begins a new accumulation.

PROCESS_SIGN_OFF

Allows a privileged service transaction program (the *Sign-Off TP*) to instruct the LU to receive and process one or more Sign-Off GDS variables.

PROCESS_SIGN_OFF	<i>Option No.</i> [219]
Supplied Parameters:	
RESOURCE (<i>variable</i>)	
Returned Parameters:	
RETURN_CODE (<i>variable</i>)	

Supplied Parameters:

RESOURCE specifies the variable containing the resource ID of the conversation that started the Sign-Off transaction program.

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- OK
- PROGRAM_PARAMETER_CHECK, for the following reason:
 - The program issuing this verb does not have display privilege.
- RESOURCE_FAILURE_NO_RETRY

Notes:

1. This verb is issued by an SNA service transaction program called the "Sign-Off TP," identified with the TP name of hex 06F3F0F0. The Sign-Off TP is invoked at the target LU as a result of a SIGNOFF verb being issued at the source LU. The Sign-Off TP then issues this verb in order to initiate the target LU's processing of the sign-off request sent by the source LU.
2. After the issuance of the PROCESS_SIGN_OFF verb, the LU is responsible for receiving and parsing one or more Sign-Off GDS variables, removing the corresponding entries from its signed-on-to list, responding with CONFIRMED if appropriate, and ending the conversation with DEALLOCATE TYPE(LOCAL).
3. If there is data in the Sign-Off GDS variable that does not correspond to an entry in this LU's signed-on-to list or is otherwise not understood by the

PROCESS_SIGN_OFF

receiving LU, the operator is informed, the variable is ignored, and CONFIRMED is still sent if appropriate.

SIGNOFF

Allows a privileged program to remove list entries meeting specified criteria from the LU's signed-on lists. The SIGNOFF verb is the signed-on list maintenance tool for use by the control operator.

SIGNOFF	<i>Option No.</i> [219,220]
Supplied Parameters: LIST (<i>ALL_SIGNED_ON_LISTS</i> (<i>IMMEDIATE</i> (<i>NO</i>) (<i>YES</i>))) (<i>SIGNED_ON_TO_LIST</i>) (<i>SIGNED_ON_FROM_LIST</i> (<i>IMMEDIATE</i> (<i>NO</i>) (<i>YES</i>))) USER (<i>ALL</i>) (<i>USER_ID</i> (<i>variable1</i>, <i>variable2</i>, ... <i>variableN</i>)) PROFILE (<i>ALL</i>) (<i>PROFILE</i> (<i>variable</i>)) LU (<i>ALL</i>) (<i>LU_NAME</i> (<i>variable</i>)) ADDITIONAL_CRITERIA (<i>NONE</i>) (<i>REMOVE_INACTIVE</i> (<i>variable</i>))	
Returned Parameters: RETURN_CODE (<i>variable</i>)	

Supplied Parameters:

LIST specifies the name of the signed-on list from which list entries are to be removed.

- **SIGNED_ON_TO_LIST** specifies the list of users who are signed on to remote LUs from the local LU.
- **SIGNED_ON_FROM_LIST** specifies the list of remote users who are signed on to the local LU.
 - **IMMEDIATE** specifies when the list entries are to be removed from the local signed-on-from list.
 - **NO** specifies that the SIGNOFF request is to be followed by a DEALLOCATE TYPE(CONFIRM). The LU is to wait for a response before removing the list entries from the signed-on-from list.

SIGNOFF

- **YES** specifies that the SIGNOFF request is to remove the list entries from the local signed-on-from list immediately. A DEALLOCATE TYPE(FLUSH) is then to be issued, unconditionally ending the conversation.
- **ALL_SIGNED_ON_LISTS** specifies both the SIGNED_ON_TO_LIST and SIGNED_ON_FROM_LIST.
 - **IMMEDIATE** specifies when the list entries are to be removed from the local signed-on-from list.
 - **NO** specifies that the SIGNOFF request is to be followed by a DEALLOCATE TYPE(CONFIRM). The LU is to wait for a response before removing list entries from the local signed-on-from list.
 - **YES** specifies that the list entries are to be removed immediately from the local signed-on-from list. A DEALLOCATE TYPE(FLUSH) is then issued, unconditionally ending the conversation.

USER specifies the user IDs of the list entries to be removed.

- **ALL** specifies that all user IDs are to be used.
- **USER_ID** specifies the variables containing the user IDs of the list entries to be removed.

PROFILE specifies the profile of the list entries to be removed.

- **ALL** specifies that all profiles are to be used.
- **PROFILE** specifies the variable containing the profile of the list entries to be removed. A variable set to the null string specifies the entries that lack a profile.

LU specifies the LU names of the list entries to be removed.

- **ALL** specifies that all LU names are to be used.
- **LU_NAME** specifies the variable containing the LU name for the list entries to be removed.

ADDITIONAL_CRITERIA specifies other conditions to be selected for removal of entries from the signed-on lists.

- **NONE** specifies that there are no additional criteria.
- **REMOVE_INACTIVE** specifies the variable containing the time period beyond which the list entries that have been inactive (that is, had active conversations) are to be removed from the specified lists. These list entries must also satisfy any other specified removal criteria (user, profile, LU).

Returned Parameters:

RETURN_CODE specifies the variable in which the result of the verb execution is indicated to the local program.

- **OK**
- **PARAMETER_ERROR**
- **PROGRAM_PARAMETER_CHECK**, for the following reason:
 - The program issuing this verb does not have display privilege.

- RESOURCE_FAILURE_RETRY
- RESOURCE_FAILURE_NO_RETRY

Notes:

1. Any entry from a signed-on list that is to be removed must meet all criteria specified on the verb for USER, PROFILE, LU, and ADDITIONAL_CRITERIA. Failure to meet the qualifications for any one of the specified parameters will result in the entry being left on the signed-on list.

SIGNOFF

Part 4. Appendixes

Appendix A. Base and Option Sets for Product Support	A-1
LU 6.2 Functions	A-1
LU 6.2 Verbs	A-2
LU 6.2 Verb Parameters	A-3
LU 6.2 Implementation Choices and Using TP Choices	A-3
List of Base Set Functions	A-6
List of Option Sets	A-8
LU 6.2 Option Sets for Conversation Verbs	A-21
LU 6.2 Option Sets for Conversation and Control-Operator Verbs, Jointly.	A-22
LU 6.2 Option Sets for Control-Operator Verbs	A-22
Support for Base and Option Sets	A-23
Support for Conversation Verbs	A-25
Mapped Conversation Verbs and Parameters	A-25
Type-Independent Conversation Verbs and Parameters	A-31
Basic Conversation Verbs and Parameters	A-33
Conversation Return Codes and What-Received Indications	A-39
Support for Control-Operator Verbs	A-42
Control-Operator Verbs and Parameters for CNOS	A-42
Control-Operator Verbs and Parameters for Session Control	A-44
Control-Operator Verbs and Parameters for LU Definition	A-45
Control-Operator Verbs and Parameters for Miscellaneous Verbs	A-51
Control-Operator Return Codes	A-52
Notes on Implementation Details	A-53
 Appendix B. Examples Using Basic Conversation Verbs	 B-1
Nonblocking Examples	B-31
 Appendix C. Symbol String Conventions	 C-1
Symbol String Type	C-1
Symbol String Length	C-4
Symbol String Processing Conventions for Symbol Strings A, 1134, AE, GR, and DB	C-5
Uppercase and Lowercase	C-6
Leading and Trailing Blanks	C-6
Imbedded Blanks	C-6
Imbedded Special Characters	C-6
Invalid Characters	C-6
Symbol String Processing with Symbol String G	C-6
Uppercase and Lowercase	C-6
Leading and Trailing Blanks	C-6
Imbedded Blanks	C-6
Imbedded Special Characters	C-7
Invalid Characters	C-7
Symbol String Conversions	C-7
 Appendix D. List of SNA Service Transaction Programs	 D-1
SNA Service Transaction Program Names	D-1
Scheduler	D-2

Appendixes

Queue	D-2
DL/1	D-2
Change Number of Sessions	D-2
Resynchronization	D-2
Security	D-2
Distributed Data Management	D-2
Distributed Relational Database Architecture	D-3
Document Interchange Architecture	D-3
SNA Distribution Services	D-3
APPN	D-3
Management Services	D-4
File Services	D-4
Product-Oriented	D-4
 Appendix E. Conversation State Matrices	 E-1
Conversation State Matrices	E-1
Half-Duplex Conversation State Matrices	E-3
Full-Duplex Conversation State Matrices	E-13
Nonblocking State Matrices	E-16
 Appendix F. Conversation Return Codes	 F-1
Purpose of Conversation Return Code	F-1
List of Conversation Return Codes and Subcodes	F-1
Correlation of Conversation Verbs and Return Codes	F-17
 Appendix G. Control-Operator Return Codes	 G-1
Purpose of Control Operator Return Code	G-1
List of Control Operator Return Codes and Subcodes	G-1
Correlation of Control Operator Verbs and Return Codes	G-4

Appendix A. Base and Option Sets for Product Support

This appendix identifies LU 6.2 functions and separates them into:

- one **base set** of functions, which **MUST** be implemented, and
- multiple **option sets** of functions, each of which **MAY** be implemented or omitted.

By using the option-set numbering introduced in this appendix, any LU 6.2 product implementation can precisely identify its support of LU 6.2 functions. Each option set must be completely implemented and some option sets have other option sets as prerequisites. See Figure A-5 on page A-21 and Figure A-7 on page A-22.

LU 6.2 Functions

The LU 6.2 functions available to a transaction program are described in this book by means of verbs and their **supplied and returned parameters**.

Some functions are in the base set of LU 6.2 functions; the remainder are in the option sets.

Some functions involve one or more verbs, while other functions involve just a parameter (or a few parameters) on one or more verbs. In other words, some functions are **verb-oriented** while others are **parameter-oriented**. For sets of optional functions not in the base set, the option-set numbers are assigned in SNA¹ in such a way as to separate the verb-oriented from the parameter-oriented, as shown in Figure A-1 on page A-2.

¹ In past editions of this book, the option set names were fixed, but the option set numbers assigned varied in each edition, being intended only for identification purposes within the edition; inconsistencies existed in correlating numbers and names from edition to edition.

In the future, IBM implementations will use the new numbering scheme (with 3-digit option set numbers) within their own documentation to characterize their support. Specification of implementation support for option sets in previous documentation was unambiguous only when option set names were used.

Product Base and Option Sets

	Option Sets		Base Set
	Conversation Verbs	Control-Operator Verbs	Conversation and Control-Operator Verbs
Verb-Oriented	1xx	5xx	No numbers in base set
Parameter-Oriented	2xx	6xx	
Verb-Oriented or Parameter-Oriented affecting both type of verbs	4xx		

Figure A-1. LU 6.2 Functions and Option-Set Numbering

LU 6.2 Verbs

The LU 6.2 conversation and control-operator verbs are listed in Figure A-4 on page A-5 as being either in the base set of functions or in an option set. Similar tables for verb parameters appear later in this appendix (see pages A-25 to A-52).

Note: Notice, in Figure A-4 on page A-5, the distinction between:

1. the **BASE** set of LU 6.2 functions and
2. the **BASIC** conversation verbs.

Only a subset of the BASIC conversation verbs is in the BASE set of LU 6.2 functions.

Verbs are either in the base set or not and, likewise, verb parameters are either in the base set or not. Parameters on base-set verbs are either in the base set of LU 6.2 functions or in the option sets. Parameters on option-set verbs are always in the option sets. For convenience, verbs and parameters are sometimes called *base* or *optional* verbs and parameters, depending upon whether they are in the base set or in option sets, respectively. Figure A-2 on page A-3 illustrates the general relationships.

[illegible]

Figure A-2. LU 6.2 Verbs, Verb Parmeters, and Option Sets

Note: This book describes only the conversation protocol boundary (conversation and control-operator verbs) of LU 6.2; see the *SNA LU 6.2 Reference: Peer Protocols* for descriptions of the internal logic and protocols of LU 6.2 and how they relate to the LU 6.2 base and option sets. See *Systems Network Architecture Formats* for the formats used by LU 6.2.

LU 6.2 Verb Parameters

All the supplied and returned verb parameters are listed in the tables on pages A-25 to A-52, where they are identified as being in the base set, in one option set, or in multiple option sets.

The supplied parameters include those parameters specified, defaulted, or optionally absent without default (for example, parameters found in option sets not supported within the implementation would be absent with a default value).

The returned parameters include:

1. the **return codes** of the RETURN_CODE parameter, defined for all verbs, and
2. the **what-received indications** of the WHAT_RECEIVED parameter, defined for the MC_RECEIVE_AND_WAIT, MC_RECEIVE_IMMEDIATE, RECEIVE AND WAIT, and RECEIVE IMMEDIATE verbs.

LU 6.2 Implementation Choices and Using TP Choices

An LU 6.2 implementation is free to choose from any of the LU 6.2 options sets available for implementation but it must implement the LU 6.2 base set of functions. The LU 6.2 implementation may choose to implement only the base set of functions and none of the option sets. If any option set is implemented, it must be completely implemented just as the base set must be completely implemented.

The using transaction program (TP) is free to choose from any of the verbs and verb parameters that are available from its local LU 6.2 implementation.

The relationship between LU 6.2 implementation choices and end-user TP choices is illustrated by the example in Figure A-3 on page A-4: implementations choose option sets to support; using TPs choose verbs and verb parameters from those available in their local implementations.

Product Base and Option Sets

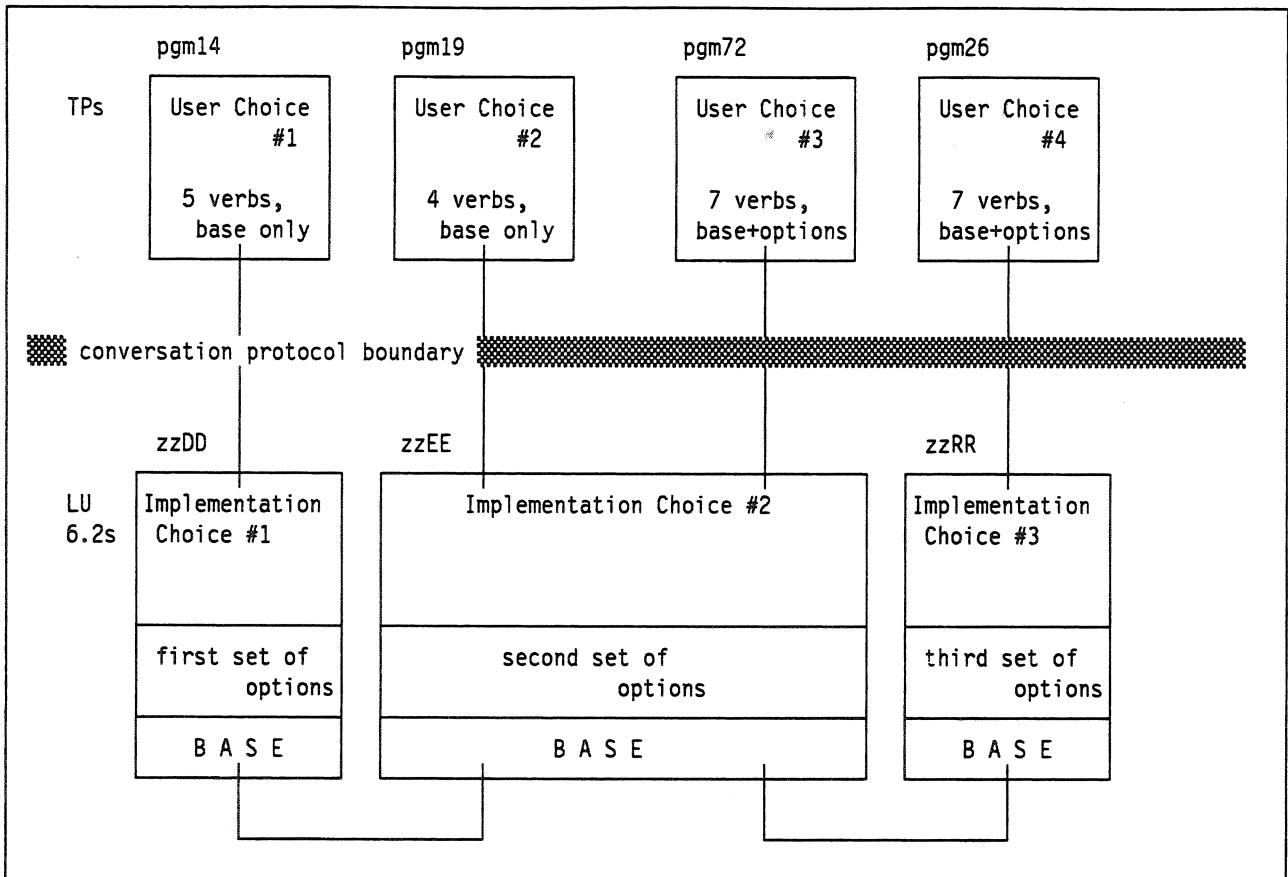


Figure A-3. Example of LU 6.2 Implementation Choices and Using TP Choices

In Figure A-3, TPN=pgm14 at LUNAME=zzDD is communicating with TPN=pgm19 at LUNAME=zzEE using only verbs in the LU 6.2 base set of functions while TPN=pgm26 at LUNAME=zzRR is communicating with TPN=pgm72 at LUNAME=zzEE using verbs both from the LU 6.2 base set and from some option sets of functions. In this example, the three LUs are shown to have three different LU 6.2 implementations (such as by three different products) but any two transaction programs at any two LU locations (or at the same LU location) can communicate without restrictions using the LU 6.2 base set of functions.

			Verbs in the BASE Set	Verbs in the OPTION Sets (see Note 1)
Conversation Protocol Boundary of LU 6.2	Conver- sation Verbs	Basic	ALLOCATE CONFIRM CONFIRMED DEALLOCATE GET_ATTRIBUTES RECEIVE_AND_WAIT REQUEST_TO_SEND SEND_DATA SEND_ERROR	FLUSH (101) POST_ON_RECEIPT (103/104) PREPARE_FOR_SYNCPT (111) PREPARE_TO_RECEIVE (105) RECEIVE_IMMEDIATE (106) TEST (103) RECEIVE_ EXPEDITED_DATA (112) SEND_ EXPEDITED_DATA (112)
		Mapped	as above with MC prefix, minus MC_GET_ATTRIBUTES	as above with MC prefix, plus MC_GET_ATTRIBUTES (102)
		Type- independent	GET_TP_PROPERTIES	BACKOUT (108) GET_TYPE (110) SYNCPT (108) WAIT (104) WAIT_FOR_COMPLETION (113)
	Control- Operator Verbs	Change Number of Sessions	INITIALIZE_SESSION_LIMIT PROCESS_SESSION_LIMIT RESET_SESSION_LIMIT	CHANGE_SESSION_LIMIT (501)
		Session Control		ACTIVATE_SESSION (502) DEACTIVATE_ CONVERSATION_GROUP (401) DEACTIVATE_SESSION (504)
		LU Definition		DEFINE_LOCAL_LU (505) DEFINE_MODE (505) DEFINE_REMOTE_LU (505) DEFINE_TP (505) DELETE (505) DISPLAY_LOCAL_LU (505) DISPLAY_MODE (505) DISPLAY_REMOTE_LU (505) DISPLAY_TP (505)
		Miscellane- ous		DISPLAY_SIGNED_ON_LIST (219,220) SIGNOFF (219,220) PROCESS_SIGNOFF (219,220)

Figure A-4. LU 6.2 Verbs and Base and Option Sets

Product Base and Option Sets

Notes:

1. The numbers in parentheses in this column indicate the option sets in which the verbs appear.
2. Five verbs (shown only in the option sets column) appear both in the base set and in an option set (in the base set for remote support and in an option set for local support). These verbs are the (MC_)FLUSH and (MC_)PREPARE_TO_RECEIVE conversation verbs together with the CHANGE_SESSION_LIMIT, ACTIVATE_SESSION, and DEACTIVATE_SESSION control-operator verbs.

List of Base Set Functions

The verb-oriented functions (that is, functions available as verbs) that are part of the LU 6.2 base set of functions are listed in Table A-1. Each function is represented by a verb but not all the parameters and parameter values of these verbs are part of the base. Refer to the tables on pages A-25 to A-52 to identify, for each LU 6.2 verb, which parameter-oriented functions (that is functions available as parameters on verbs) are part of the base and which are part of an option set (the base is identified by a "B"; options are identified by three-digit option numbers).

Remember that parameter-oriented functions can affect multiple parameters and multiple verbs.

Table A-1 (Page 1 of 2). LU 6.2 Base Set of Functions (Verb-Oriented).

For a list of the base set of functions that are not verb-oriented, see the tables on pages A-25 to A-52.

Function Name	Function Description
Conversation Verbs:	
(MC_)ALLOCATE	Begin a conversation
(MC_)CONFIRM	Request a confirmation of "message received by remote TP"
(MC_)CONFIRMED	Send a confirmation of "message received by local TP"
(MC_)DEALLOCATE	End a conversation
GET_ATTRIBUTES	Inquire about the attributes of an LU 6.2 conversation
GET_TP_PROPERTIES	Inquire about the properties of a transaction program
(MC_)RECEIVE_AND_WAIT	Receive a message (waiting, if no message available yet)
(MC_)REQUEST_TO_SEND	Request send-control from partner transaction program
(MC_)SEND_DATA	Send a data message
(MC_)SEND_ERROR	Send an error message
Control-Operator Verbs:	

Table A-1 (Page 2 of 2). LU 6.2 Base Set of Functions (Verb-Oriented).	
For a list of the base set of functions that are not verb-oriented, see the tables on pages A-25 to A-52.	
Function Name	Function Description
INITIALIZE_SESSION_LIMIT	Initialize the number of sessions between LUs
PROCESS_SESSION_LIMIT	Process, at the remote LU, the CNOS request
RESET_SESSION_LIMIT	Reset the number of sessions between LUs to the initial value

The complete LU 6.2 base set of functions is identified jointly by:

1. verbs in Table A-1 on page A-6 and
2. verb parameters and parameter values in the tables on pages A-25 to A-52.

List of Option Sets

An LU 6.2 product may provide support for all verbs, parameters, return codes, and what-received indications, or a permitted subset of them. The permitted subsetting for LU 6.2 products is defined by means of a base set and a number of option sets. The option sets defined for the verbs, parameters, return codes, and what-received indications are shown in Table A-2.

Table A-2 (Page 1 of 13). LU 6.2 Option Sets		
Set No.	Set Name	Set Description
Conversation Verb Option Sets Related to VERBs		
101	Flush the LU's send buffer	This option set allows a program to explicitly cause the LU to flush its send buffer. This option set includes the (MC)_FLUSH verb, but just for local support, since the remote support for this verb is in the LU 6.2 base set of functions.
102	Get attributes	This option set allows a program to obtain attributes of a mapped conversation. This option set includes the MC_GET_ATTRIBUTES verb; in contrast, the GET_ATTRIBUTES verb for basic conversations is part of the LU 6.2 base set of functions. (See also option sets 108, 401, and 251, which influence the parameters that are returned.)
103	Post on receipt with test for posting	This option set allows a program to first request posting of a conversation and then to test the conversation to determine whether information is available. This option set includes the (MC_)POST_ON_RECEIPT verb and the (MC_)TEST verb. Option set 105 is a prerequisite. (See also option sets 104 and 245.)
104	Post on receipt with wait	This option set allows a program to request posting of multiple conversations and then to wait (suspend its processing) until information is available on any one of the conversations. This option set includes the (MC_)POST_ON_RECEIPT verb and the WAIT verb, a type-independent verb. Option set 105 is a prerequisite. (See also option set 103.)
105	Prepare to receive	This option set allows a program to change the conversation from send state to receive state and at the same time flush the LU's send buffer, request confirmation, or request sync point. The transaction program gets control back with the RECEIVE_AND_WAIT verb (or other similar receive verb from an option set). This option set includes the (MC_)PREPARE_TO_RECEIVE verb, but just for local support, since the remote support for this verb is in the LU 6.2 base set of functions.

Table A-2 (Page 2 of 13). LU 6.2 Option Sets		
Set No.	Set Name	Set Description
106	Receive immediate	This option set allows a program to receive whatever information is available on a conversation without having to request posting of the conversation. This option set includes the (MC_)RECEIVE_IMMEDIATE verb. Option set 105 is a prerequisite.
108	Sync point services	This option set allows a program to request sync point processing for all protected resources of the transaction. This option set includes the SYNCPT, BACKOUT, and SET_SYNCPT_OPTIONS verbs, all of which are type-independent verbs. This option set relates to the SYNC_LEVEL parameter of the (MC_)ALLOCATE, (MC_)DEALLOCATE, (MC_)GET_ATTRIBUTES, DEFINE_TP, and DISPLAY_TP verbs, to the PROTECTED_LUW_IDENTIFIER parameter of the GET_TP_PROPERTIES verb, and to the CONVERSATION_STATE parameter of the (MC_)GET_ATTRIBUTES verb.
109	Get TP name and instance identifier	This option set allows a program to request information about the name and instance identifier for the TP issuing the verb. This option set includes the OWN_TP_NAME and OWN_TP_INSTANCE parameters of the GET_TP_PROPERTIES verb, a type-independent verb in the base set of functions.
110	Get conversation type	This option set allows a program that supports both the basic conversation and mapped conversation protocol boundaries to determine which category of verbs it should use in conjunction with a resource ID. This option set includes the GET_TYPE verb, a type-independent verb.
111	Recovery from Program Errors Detected During Sync Point	This option set allows a program to issue the (MC_)PREPARE_FOR_SYNCPT verb to initiate a sync point operation on a conversation. If a program error return code is received after this verb is issued, the program can attempt to fix the problem and retry the sync point operation. This contrasts to the SYNCPT verb, which backs out the transaction if a program error is detected during the sync point operation. This option set includes the (MC_)PREPARE_FOR_SYNCPT verb and affects the return code of the BACKOUT verb. Option set 108 is a prerequisite.

Product Base and Option Sets

Table A-2 (Page 3 of 13). LU 6.2 Option Sets		
Set No.	Set Name	Set Description
112	Full-duplex conversations and expedited data	This option set allows a program to establish a full-duplex conversation with a partner transaction program that also supports full-duplex conversations, and to send and receive user-defined expedited data. The expedited data feature is provided for both full-duplex and half-duplex conversations. This option set includes the (MC_)SEND_EXPEDITED_DATA and (MC_)RECEIVE_EXPEDITED_DATA verbs, and relates to the TYPE parameter of the (MC_)ALLOCATE verb. Option sets 101, 110, 113, and 247 are prerequisites.
113	Nonblocking support	This option set allows a program to issue a verb and obtain control back prior to the completion of the verb. The program can have multiple verbs outstanding and then test or wait for their completion. The number of outstanding verbs allowed in each LU 6.2 queue is a product choice. This option set includes an optional parameter WAIT_OBJECT on the verbs whose execution could otherwise be blocked and the WAIT_FOR_COMPLETION verb, a type-independent verb.
Conversation Verb Option Sets Related to VERB PARAMETERS		
201	Queued allocation of a contention winner session	This option set allows a local program to allocate a conversation to a remote program on a session for which the local LU must be the conwinner and for which the local program will wait. This option set relates to the RETURN_CONTROL parameter of the (MC_)ALLOCATE verb.
203	Immediate allocation of a session	This option set allows a program to allocate a contention-winner session only if one is immediately available; otherwise, the allocation is unsuccessful. This option set relates to the RETURN_CONTROL parameter of the (MC_)ALLOCATE verb.
204	Conversations between programs located at the same LU	This option set allows a local program to allocate a conversation to a program located at the same LU as the local program. This option set relates to the LU_NAME parameter of the (MC_)ALLOCATE verb.

Table A-2 (Page 4 of 13). LU 6.2 Option Sets		
Set No.	Set Name	Set Description
205	Queued allocation for when session free	This option set allows a local program to allocate a conversation to a remote program on a session for which the local program is willing to wait for the duration of a session activation, if necessary. If session activation cannot be accomplished by the LU 6.2, the local program will receive control back following the activation failure. This option set relates to the RETURN_CONTROL parameter of the (MC_)ALLOCATE verb.
211	Session-level LU-LU verification	This option set allows a program or an operator to designate the LU-LU passwords, associated with remote LUs, that the local LU uses to verify the identity of a remote LU at session activation time. This option set relates to the LU_LU_PASSWORD of the DEFINE_REMOTE_LU verb. Note: This option set is a prerequisite for every other security-related option set.
212	User ID verification	This option set allows a program or an operator to designate the user IDs and associated passwords that the local LU uses to verify the identity of a user ID carried on allocation requests it receives, and to designate the remote LUs that are permitted to send to the local LU allocation requests carrying a user ID and either a password or an already-verified indication. This option set also allows the program allocating a conversation to specify that the allocation request carry access security information determined by the local LU. This option set relates to the SECURITY parameter of the (MC_)ALLOCATE, DEFINE_LOCAL_LU, and DISPLAY_LOCAL_LU verbs and to the SECURITY_USERID parameter of the GET_TP_PROPERTIES verb. It also relates to the SECURITY_REQUIRED parameter of the DEFINE_TP and DISPLAY_TP verbs, the SECURITY_ACCEPTANCE parameter of the DEFINE_REMOTE_LU verb, and to the SECURITY_ACCEPTANCE_LOCAL_LU and SECURITY_ACCEPTANCE_REMOTE_LU parameters of the DISPLAY_REMOTE_LU verb. Option set 211 is a prerequisite.
213	Program-supplied user ID and password	This option set allows the program allocating a conversation to supply the user ID and password to be sent on the allocation request. This option set relates to the SECURITY parameter of the (MC_)ALLOCATE verb. Option set 211 is a prerequisite.

Product Base and Option Sets

Table A-2 (Page 5 of 13). LU 6.2 Option Sets		
Set No.	Set Name	Set Description
214	User ID authorization	This option set allows a program or an operator to designate the user IDs that are authorized access to specific resources of the LU, such as transaction programs. This option set relates to the SECURITY parameter of the (MC_)ALLOCATE verb and to the SECURITY_ACCESS and SECURITY_REQUIRED parameters of the DEFINE_TP and DISPLAY_TP verbs. Option set 212 is a prerequisite.
215	Profile verification and authorization	This option set allows a program or an operator to designate the profiles that the local LU uses to verify a profile carried on allocation requests it receives, and to designate the profiles that are authorized access to specific resources of the LU, such as transaction programs. This option set relates to the SECURITY parameter of the (MC_)ALLOCATE and DEFINE_LOCAL_LU verbs, to the SECURITY_PROFILE parameter of the GET_TP_PROPERTIES verb, and to the SECURITY_ACCESS and SECURITY_REQUIRED parameters of the DEFINE_TP and DISPLAY_TP verbs. Option set 212 is a prerequisite.
216	Origin LU authorization	This option set allows a program or an operator to designate the user IDs and the remote LUs, from which allocation requests carrying those user IDs are received, that are authorized access to specific resources of the LU, such as transaction programs. This option set relates to the SECURITY parameter of the (MC_)ALLOCATE verb and to the SECURITY_ACCESS and SECURITY_REQUIRED parameters of the DEFINE_TP and DISPLAY_TP verbs. Option set 214 is a prerequisite.
217	Profile passthrough	This option set allows the program allocating a conversation to specify that the allocation request carry the profile received on the request that started the program. This option set relates to the SECURITY parameter of the (MC_)ALLOCATE verb. Option set 212 is a prerequisite.
218	Program-supplied profile	This option set allows the program allocating a conversation to supply the profile to be sent on the allocation request. This option set relates to the SECURITY parameter of the (MC_)ALLOCATE verb. Option set 213 is a prerequisite.

Table A-2 (Page 6 of 13). LU 6.2 Option Sets		
Set No.	Set Name	Set Description
219	Send persistent verification	This option set allows a program to allocate consecutive conversations during which the conversation-level security information, once verified, remains verified for specific combinations of user IDs, profiles, and partner LUs. This option set relates to the SECURITY parameter of the (MC_)ALLOCATE verb. The parameters of the DISPLAY_SIGNED_ON_LIST, SIGNOFF, and PROCESS_SIGNOFF verbs could also be affected by this option set. Option set 211 is a prerequisite.
220	Receive persistent verification	This option set allows a program or an operator to designate the remote LUs from which the local LU will accept consecutive allocation requests that, once verified, remain verified for specific combinations of user IDs, profiles, and partner LUs. This option set relates to the SECURITY parameter of the (MC_)ALLOCATE verb and to the SECURITY_ACCEPTANCE parameter of the DEFINE_REMOTE_LU verb. The parameters of the DISPLAY_SIGNED_ON_LIST, SIGNOFF, and PROCESS_SIGNOFF verbs could also be affected by this option set. Option set 212 is a prerequisite.
221	Send Sign-On/Change-Password	This option set allows the local LU to send a "Sign-On/Change-Password" request to the remote LU. If option set 219 is in use, subsequent Attaches for this user can flow as "signed-on."
222	Receive Sign-On/Change-Password	This option set allows the LU to receive a "Sign-On/Change-Password" and schedule the "Sign-On/Change-Password TP." The user's password may be changed as a result. If option set 220 is in use, subsequent Attaches for this user can flow as "signed-on."
241	Send PIP data	This option set allows the local program allocating a conversation to provide the remote program with initialization parameters. This option set is only for send support. This option set relates to the PIP parameter of the (MC_)ALLOCATE verb and to the PIP parameter of the DEFINE_TP and DISPLAY_TP verbs.
242	Receive PIP data	This option set allows the local program to receive from the remote program allocating a conversation some initialization parameters. This option set is only for receive support. This option set relates to the PIP parameter of the (MC_)ALLOCATE verb at the remote LU and to the PIP parameter of the DEFINE_TP and DISPLAY_TP verbs at the local LU.

Product Base and Option Sets

Table A-2 (Page 7 of 13). LU 6.2 Option Sets		
Set No.	Set Name	Set Description
243	Accounting	This option set allows an LU implementation to generate and send both a logical-unit-of-work (LUW) identifier and a conversation correlator (CC) to the remote LU. This option set provides the LUW and CC identifiers to be used, along with other information, for accounting purposes. The LU generates the identifiers. This option set relates to the CONVERSATION_CORRELATOR parameter on the (MC_)GET_ATTRIBUTES verb and to the LUW_IDENTIFIER parameter on the GET_TP_PROPERTIES verb. See also option set 251, which allows the TP to retrieve these identifiers.
244	Long locks	This option set allows a program to perform the prepare-to-receive function and request confirmation, and resume processing when information, such as data or conversation status, is received from the remote program following an affirmative reply. This option set relates to the LOCKS parameter of the (MC_)PREPARE_TO_RECEIVE verb. Option set 105 is a prerequisite.
245	Test for request-to-send received	This option set allows a program to test whether a request-to-send notification has been received on a conversation, for example following sync point processing. This option set relates to the TEST parameter of the (MC_)TEST verb. (See also option set 103.)
246	Data mapping	This option set allows a program using mapped conversations to request mapping of the data by the local and remote LUs. This option set relates to the MAP_NAME parameter of the MC_SEND_DATA, MC_RECEIVE_AND_WAIT, MC_RECEIVE_IMMEDIATE, and DEFINE_LOCAL_LU verbs and to the DATA_MAPPING parameter of the DEFINE_TP and DISPLAY_TP verbs.
247	User control data	This option set allows programs to send and receive data records containing user control data. The user control data has meaning only to the application programs. This option set relates, for mapped conversations only, to the USER_CONTROL_DATA parameter of the MC_SEND_DATA verb.

Table A-2 (Page 8 of 13). LU 6.2 Option Sets		
Set No.	Set Name	Set Description
249	Vote Read-Only Response to a Sync Point Operation	This option set improves performance of sync point operations by allowing the local LU to vote read-only when none of the protected resources in its part of the distributed transaction have been changed. This option set includes the <code>VOTE_READ_ONLY_PERMITTED</code> parameter of the <code>SET_SYNCPT_OPTIONS</code> verb, and affects the return code to the <code>SYNCPT</code> verb. Option set 108 is a prerequisite.
250	Presume Abort Performance Improvements for Sync Point	This option set improves performance of sync point operations by using presume abort algorithms and by making use of information about the distributed transaction supplied by the programmer. This option set includes the <code>OK_TO_LEAVE_OUT</code> and <code>SELECT_LAST_AGENT</code> parameters of the <code>SET_SYNCPT_OPTIONS</code> verb. Option set 108 and 249 are prerequisites.
251	Extract Transaction and Conversation Identification Information	This option set allows the TP to retrieve the information identifying the transaction and the conversations it is using. This option set relates to the <code>LUW_IDENTIFIER</code> and <code>PROTECTED_LUW_IDENTIFIER</code> parameters of the <code>GET_TP_PROPERTIES</code> verb and to the <code>CONVERSATION_CORRELATOR</code> and <code>SESSION_ID</code> parameters of the <code>(MC_)GET_ATTRIBUTES</code> verb. Option sets 109 and 243 are prerequisites.
290	Logging of data in a system log	This option set allows a program to record error information in the system's error log. This option can be used on basic conversations only and relates to the <code>LOG_DATA</code> parameter of the <code>SEND_ERROR</code> and <code>DEALLOCATE</code> verbs.
291	Mapped conversation LU services component	This option set allows implementation of a mapped conversation LU services component program, which processes mapped conversation verbs. This option can be used on basic conversations only and relates to the <code>TYPE</code> parameter of the <code>ALLOCATE</code> , <code>DEALLOCATE</code> , and <code>SEND_ERROR</code> verbs.

Product Base and Option Sets

Table A-2 (Page 9 of 13). LU 6.2 Option Sets		
Set No.	Set Name	Set Description
Conversation and Control-Operator Verb Option Sets Related to VERBs		
401	Reliable One-way Brackets	This option set supplies the support necessary to reliably send one-way brackets (a conversation where communication is in one direction.). In particular, this option set allows a program to allocate a conversation to a remote program on a particular session that is uniquely identified by a conversation group ID. It also allows for a particular transaction program always to be started whenever a session has been deactivated. This combination of starting a transaction program on a session deactivation and the unique conversation group ID provides the tools necessary to allow the user to be able to identify the potential failed one-way bracket and take appropriate action. This option set relates to the RETURN_CONTROL and CONVERSATION_GROUP_ID parameters of the (MC_)ALLOCATE verb, to the CONVERSATION_GROUP_ID parameter of the (MC_)GET_ATTRIBUTES verb, to the DEACTIVATE_CONVERSATION_GROUP verb, and the SESSION_DEACTIVATED_TP_NAME parameter of the DEFINE_MODE and DISPLAY_MODE verbs. (See also option set 102.)
Control-Operator Verb Option Sets Related to VERBs		
501	CHANGE_SESSION_LIMIT verb	This option set allows a program or an operator at the source LU to request a change in the (LU,mode) session limit from one nonzero value to another, or a change in the minimum number of contention-winner sessions for the source LU or target LU. This option set includes the CHANGE_SESSION_LIMIT verb, but just for local support, since the remote support for this verb is in the LU 6.2 base set of functions.
502	ACTIVATE_SESSION verb	This option set allows a program or an operator to activate LU-LU sessions. This option set includes the ACTIVATE_SESSION verb, but just for local support, since the remote support for this verb is in the LU 6.2 base set of functions. This option set relates to the PRIVILEGE parameter of the DEFINE_TP verb.

Table A-2 (Page 10 of 13). LU 6.2 Option Sets		
Set No.	Set Name	Set Description
504	DEACTIVATE_SESSION verb	This option set allows a program or an operator to deactivate LU-LU sessions. This option set includes the DEACTIVATE_SESSION verb, but just for local support, since the remote support for this verb is in the LU 6.2 base set of functions. This option set relates to the PRIVILEGE parameter of the DEFINE_TP verb. See also option set 251.
505	LU-definition verbs	This option set allows a program or an operator to specify the operating parameters of its LU. Within this option set, the individual operating parameters that a product supports and makes accessible to the program or operator are product-dependent. This option set includes the DEFINE_LOCAL_LU, DEFINE_MODE, DEFINE_REMOTE_LU, DEFINE_TP, DELETE, DISPLAY_LOCAL_LU, DISPLAY_MODE, DISPLAY_REMOTE_LU, DISPLAY_SESSION, and DISPLAY_TP verbs.
Control-Operator Verb Option Sets Related to VERB PARAMETERS		
601	MIN_CONWINNERS_TARGET parameter	This option set allows a program or an operator at the source LU to request a nonzero value for the target LU's minimum number of contention-winner sessions. This option set relates to the MIN_CONWINNERS_TARGET parameter of the INITIALIZE_SESSION_LIMIT (Base) verb and the optional CHANGE_SESSION_LIMIT (option 501) verb. If this parameter is supported on the INITIALIZE_SESSION_LIMIT verb and option set 501 is also supported, then this parameter must also be supported on the CHANGE_SESSION_LIMIT verb.
602	RESPONSIBLE(TARGET) parameter	This option set allows a program or an operator at the source LU to request that the target LU be responsible for session deactivations when the verb requires a decrease in the number of active sessions. This option set relates to the RESPONSIBLE parameter of the RESET_SESSION_LIMIT (Base) verb and the optional CHANGE_SESSION_LIMIT (option 501) verb. If this parameter is supported on the RESET_SESSION_LIMIT verb and option set 501 is also supported, then this parameter must also be supported on the CHANGE_SESSION_LIMIT verb.

Product Base and Option Sets

Table A-2 (Page 11 of 13). LU 6.2 Option Sets		
Set No.	Set Name	Set Description
603	DRAIN_TARGET(NO) parameter	This option set allows a program or an operator at the source LU to prevent the target LU from draining its allocation requests as a result of resetting the (LU,mode) session limit to 0. This option set relates to the DRAIN parameter of the RESET_SESSION_LIMIT verb.
604	FORCE parameter	This option set allows a program or an operator to specify that the (LU,mode) session limit is to be reset to 0 even if the CNOS exchange between the source LU and target LU is unsuccessful. This option set relates to the FORCE parameter of the RESET_SESSION_LIMIT verb.
605	LU-LU session limit	This option set allows a program or an operator to specify the LU-LU session limit. This option set relates to the LU_SESSION_LIMIT parameter of the DEFINE_LOCAL_LU verb.
606	Locally known LU names	This option set allows a program or an operator to specify the locally known names of remote LUs. This option set relates to the LOCALLY_KNOWN_LU_NAME parameter of the DEFINE_REMOTE_LU and DISPLAY_REMOTE_LU verbs. The locally known LU name value may be used in the LU_NAME parameter of the (MC_)ALLOCATE verb and other verbs having the LU_NAME parameter. (See the "LU_NAME parameter" entry in the index for a list of verbs that have the LU_NAME parameter.)
607	Uninterpreted LU names	This option set allows a program or an operator to specify the uninterpreted names of remote LUs. This option set relates to the UNINTERPRETED_LU_NAME parameter of the DEFINE_REMOTE_LU and DISPLAY_REMOTE_LU verbs. The uninterpreted LU name value may be used in the LU_NAME parameter of the (MC_)ALLOCATE verb and other verbs having the LU_NAME parameter. (See the "LU_NAME parameter" entry in the index.)
608	Single-session reinitiation	This option set allows a program or an operator to specify the responsibility for reinitiation of single sessions to remote LUs. This option set relates to the SINGLE_SESSION_REINITIATION parameter of the DEFINE_MODE and DISPLAY_MODE verbs.

Table A-2 (Page 12 of 13). LU 6.2 Option Sets		
Set No.	Set Name	Set Description
610	Maximum RU size bounds	This option set allows a program or an operator to specify the lower and upper bounds for the maximum RU sizes on sessions within an (LU,mode) group. This option set relates to the SEND_MAX_RU_SIZE_LOWER_BOUND, SEND_MAX_RU_SIZE_UPPER_BOUND, RECEIVE_MAX_RU_SIZE_LOWER_BOUND, and RECEIVE_MAX_RU_SIZE_UPPER_BOUND parameters of the DEFINE_MODE and DISPLAY_MODE verbs.
611	Session-level mandatory cryptography	This option set allows a program or an operator to specify that session-level mandatory cryptography is to be used on sessions within an (LU,mode) group. This option set relates to the SESSION_LEVEL_CRYPTOGRAPHY parameter of the DEFINE_MODE and DISPLAY_MODE verbs.
612	Contention winner automatic activation limit	This option set allows a program or an operator to specify the limit for automatically activating contention-winner sessions within an (LU,mode) group. This option set relates to the CONWINNER_AUTO_ACTIVATE_LIMIT parameter of the DEFINE_MODE and DISPLAY_MODE verbs.
613	Local maximum (LU,mode) session limit	This option set allows a program or an operator to specify the maximum value that can be used during CNOS processing of the (LU,mode) session limit. This option set relates to the LOCAL_MAX_SESSION_LIMIT parameter of the DEFINE_MODE and DISPLAY_MODE verbs.
616	CPSVCMG mode name support	This option set allows a program or an operator to support the SNA-defined CPSVCMG mode name that is used for exchanging control messages between source LU and target LU. This option set relates to the MODE_NAME parameter of the INITIALIZE_SESSION_LIMIT, RESET_SESSION_LIMIT, and ACTIVATE_SESSION verbs.
617	Session-level selective cryptography	This option set is an extension to option set 611 and allows for the TP to send both encrypted and unencrypted data on the same session. This option set relates to the SESSION_LEVEL_CRYPTOGRAPHY parameter of the DEFINE_MODE and DISPLAY_MODE verbs and to the ENCRYPT parameter of the (MC_)SEND_DATA verb. Option set 611 is a prerequisite.

Product Base and Option Sets

Table A-2 (Page 13 of 13). LU 6.2 Option Sets		
Set No.	Set Name	Set Description
618	Length-checked compression	This option set allows transaction programs to use sessions with data compression. Length-checked compression is transparent to the transaction program. This option set relates to the DEFINE_LOCAL_LU, DEFINE_MODE, DISPLAY_LOCAL_LU, and DISPLAY_MODE verbs through the parameters LENGTH_CHECKED_COMPRESSION, INBOUND_COMPRESSION_LEVEL, OUTBOUND_COMPRESSION_LEVEL, and RLE_BEFORE_LZ.

LU 6.2 Option Sets for Conversation Verbs

The relationships among the LU 6.2 option sets for conversation verbs are shown in Figure A-5.

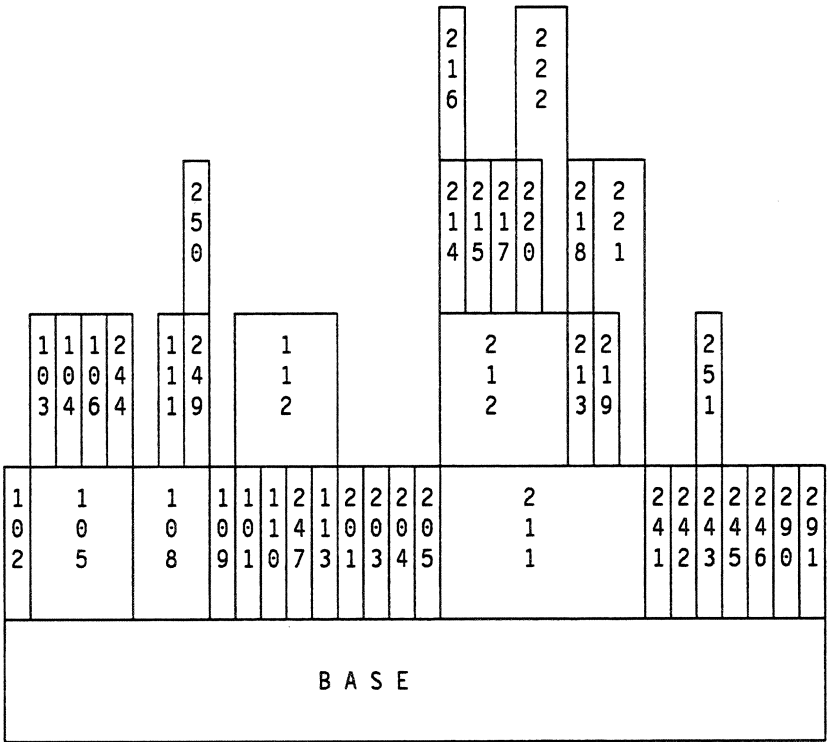


Figure A-5. LU 6.2 Option Sets and Their Prerequisites for Conversation Verbs

4 0 1	
BASE	

LU 6.2 Option Sets for Control-Operator Verbs

[illegible]

A-22 SNA Transaction Programmer's Reference Manual for LU Type 6.2

Support for Base and Option Sets

The tables on pages A-25 to A-52 identify local and remote support² of the base set and option sets for the verbs, parameters, return codes, and what-received indications. Local support, either from the base set or from one or more option sets, is defined for all of these. Remote support is defined only for the verbs and parameters, as it does not apply to the return codes and what-received indications. Two hyphens (--) are shown for remote support of a verb or parameter that does not invoke remote processing.

The verbs, parameters, return codes, and what-received indications belonging to the base set are identified by "B" in the local-support or remote-support column. Those belonging to an option set are identified by the number of that option set.

For some of the verbs, parameters, return codes, and what-received indications, more than one option set is identified. An identification of the form "a or b" means the verb, parameter, return code, or what-received indication is supported when either option set "a" or "b" is supported. An identification of the form "a and b" means the verb, parameter, return code, or what-received indication is supported when both option sets "a" and "b" are supported.

Notes pertaining to the base and optional support of the verbs, parameters, return codes, and what-received indications are listed following the tables, beginning with "Notes on Implementation Details" on page A-53. The verbs, parameters, return codes, and what-received indications to which the notes apply include a note reference, shown as "[n]," in the local- or remote-support column. The notes explain certain implementation details, which are product-dependent.

Note: As shown in the tables for the conversation verbs and parameters, most of the option sets are optional only for local support; remote support for the verbs and parameters of these option sets is either part of the base set (indicated with "B") or is not applicable (indicated with "--"). The local program may use these conversation verbs and parameters whenever its product supports them. Use of the remaining conversation verbs and parameters—those for which remote support of an option set is shown—depends on the remote support that the remote product provides. In particular, the local program may use the verbs and parameters of the following option sets whenever its product supports them and the remote product provides the support indicated in the remote-support column:

- 108 Sync point services
- 112 Full-duplex conversations and expedited data
- 212 End-user verification
- 213 Program-supplied user ID and password
- 217 Profile passthrough
- 218 Program-supplied profile
- 219 Send persistent verification
- 241 Send PIP data
- 242 Receive PIP data

² Local support refers to information flowing only within a local LU or flowing from a local LU outbound to a remote LU; remote support refers to information flowing from a remote LU inbound to a local LU.

Product Base and Option Sets

246 Data mapping
247 User control data

Support for Conversation Verbs

Mapped Conversation Verbs and Parameters

Table A-3 (Page 1 of 6). Support for Mapped Conversation Verbs and Parameters		
Verb and Parameter	Local Support	Remote Support
MC_ALLOCATE	B	B [1]
LU_NAME(variable)	B or 204	B
MODE_NAME	B	B
TPN	B	B
TYPE		
(HALF_DUPLEX_CONVERSATION)	B	B
(FULL_DUPLEX_CONVERSATION)	112	112
RETURN_CONTROL		
(WHEN_SESSION_ALLOCATED)	B [2]	--
(IMMEDIATE)	203	--
(WHEN_CONWINNER_ALLOCATED)	201	--
(WHEN_CONVERSATION_		
GROUP_ALLOCATED)	401	--
(WHEN_SESSION_FREE)	205	--
CONVERSATION_GROUP_ID	401	--
SYNC_LEVEL		
(NONE)	B	B
(CONFIRM)	B [12]	B [12]
(SYNCPT)	108	108
SECURITY		
(NONE)	B	B
(SAME)	212, 217 [3]	212, 214, 215,
	219	216, 217 or 220
(PGM(USER_ID(variable)	213	212, 214 or 216
PASSWORD(variable)	213	212
PROFILE(variable)))	218	215 or 217
PIP		
(NO)	B	B
(YES(variable))	241 or 242 [4]	241 or 242 [4]
WAIT_OBJECT		
(BLOCKING)	B	--
(VALUE(variable))	113 [10]	--
RESOURCE	B	--
RETURN_CODE	B	--
B = base, nnn = option number, [nn] = note number on pages A-53 through A-57		

Product Base and Option Sets

Table A-3 (Page 2 of 6). Support for Mapped Conversation Verbs and Parameters		
Verb and Parameter	Local Support	Remote Support
MC_CONFIRM	B [11]	B [11]
RESOURCE	B	--
WAIT_OBJECT		
(BLOCKING)	B	--
(VALUE(variable))	113 [10]	--
RETURN_CODE	B	--
REQUEST_TO_SEND_RECEIVED	B	--
EXPEDITED_DATA_RECEIVED	112	--
MC_CONFIRMED	B [11]	B [11]
RESOURCE	B	--
WAIT_OBJECT		
(BLOCKING)	B	--
(VALUE(variable))	113 [10]	--
RETURN_CODE	B	--
MC_DEALLOCATE	B	B
RESOURCE	B	--
TYPE		
(SYNC_LEVEL)	B	B
(FLUSH)	B	B
(CONFIRM)	B [12]	B [12]
(ABEND)	B	B
(LOCAL)	B [12]	--
WAIT_OBJECT		
(BLOCKING)	B	--
(VALUE(variable))	113 [10]	--
RETURN_CODE	B	--
EXPEDITED_DATA_RECEIVED	112	--
MC_FLUSH	101	B
RESOURCE	101	--
WAIT_OBJECT		
(BLOCKING)	101	--
(VALUE(variable))	113 [10]	--
RETURN_CODE	101	--
B = base, nnn = option number, [nn] = note number on pages A-53 through A-57		

Table A-3 (Page 3 of 6). Support for Mapped Conversation Verbs and Parameters

Verb and Parameter	Local Support	Remote Support
MC_GET_ATTRIBUTES	102, 108, 212, 215, 243, 251, 401	--
RESOURCE	102, 108, 212, 215, 243, 251, 401	--
RETURN_CODE	102, 108, 212, 215, 243, 251, 401	--
PARTNER_LU_NAME	102	--
PARTNER_FULLY_QUALIFIED_LU_NAME	102	--
MODE_NAME	102	--
SYNC_LEVEL	102 and 108	--
CONVERSATION_STATE	102 and 108	--
CONVERSATION_CORRELATOR	102 and 251	--
SESSION_ID	102 and 251	--
CONVERSATION_GROUP_ID	102 and 401	--
MC_POST_ON_RECEIPT	103 or 104 [5,11]	--
RESOURCE	103 or 104	--
LENGTH	103 or 104 [14]	--
RETURN_CODE	103 or 104	--
MC_PREPARE_FOR_SYNCPT	111 [11]	--
RESOURCE	111	--
WAIT_OBJECT		
(BLOCKING)	111	--
(VALUE(variable))	113 [10]	--
RETURN_CODE	111	--
MC_PREPARE_TO_RECEIVE	105 [11]	B [11]
RESOURCE	105	--
TYPE		
(SYNC_LEVEL)	105	B
(FLUSH)	105	B
(CONFIRM)	105	B
LOCKS		
(SHORT)	105	B
(LONG)	105 and 244	B
WAIT_OBJECT		
(BLOCKING)	105	--
(VALUE(variable))	113 [10]	--
RETURN_CODE	105	--
B = base, nnn = option number, [nn] = note number on pages A-53 through A-57		

Product Base and Option Sets

Table A-3 (Page 4 of 6). Support for Mapped Conversation Verbs and Parameters		
Verb and Parameter	Local Support	Remote Support
MC_RECEIVE_AND_WAIT	B [6]	--
RESOURCE	B	--
WAIT_OBJECT		
(BLOCKING)	B	--
(VALUE(variable))	113 [10]	--
LENGTH	B [14]	--
RETURN_CODE	B	--
REQUEST_TO_SEND_RECEIVED	B [9]	--
EXPEDITED_DATA_RECEIVED	112	--
DATA	B	--
WHAT_RECEIVED	B [7]	--
MAP_NAME	246	--
MC_RECEIVE_EXPEDITED_DATA	112	112
RESOURCE	112	--
RETURN_CONTROL		
(WHEN_EXPEDITED_DATA_RECEIVED)	112	--
(IMMEDIATE)	112	--
WAIT_OBJECT		
(BLOCKING)	112	--
(VALUE(variable))	113 [10]	--
LENGTH	112	--
DATA	112	--
RETURN_CODE	112	--
REQUEST_TO_SEND_RECEIVED	112 [9]	--
EXPEDITED_DATA_RECEIVED	112	--
MC_RECEIVE_IMMEDIATE	106	--
RESOURCE	106	--
WAIT_OBJECT		
(BLOCKING)	106	--
(VALUE(variable))	113 [10]	--
LENGTH	106 [14]	--
RETURN_CODE	106	--
REQUEST_TO_SEND_RECEIVED	106 [9]	--
EXPEDITED_DATA_RECEIVED	112	--
DATA	106	--
WHAT_RECEIVED	106 [7]	--
MAP_NAME	106 and 246	--
B = base, nnn = option number, [nn] = note number on pages A-53 through A-57		

Table A-3 (Page 5 of 6). Support for Mapped Conversation Verbs and Parameters

Verb and Parameter	Local Support	Remote Support
MC_REQUEST_TO_SEND	B [8,11]	B [11]
RESOURCE	B	--
WAIT_OBJECT		
(BLOCKING)	B	--
(VALUE(variable))	113 [10]	--
RETURN_CODE	B	--
MC_SEND_DATA	B	B
RESOURCE	B	--
DATA	B	B
LENGTH	B [14]	B
MAP_NAME		
(NO)	B [15]	B
(YES(variable))	246	246
USER_CONTROL_DATA		
(NO)	B	B
(YES)	247 [16]	247 [16]
ENCRYPT		
(NO)	B	--
(YES)	617	--
WAIT_OBJECT		
(BLOCKING)	B	--
(VALUE(variable))	113 [10]	--
RETURN_CODE	B	--
REQUEST_TO_SEND_RECEIVED	B [9]	--
EXPEDITED_DATA_RECEIVED	112	--
MC_SEND_ERROR	B	B
RESOURCE	B	--
WAIT_OBJECT		
(BLOCKING)	B	--
(VALUE(variable))	113 [10]	--
RETURN_CODE	B	--
REQUEST_TO_SEND_RECEIVED	B [9]	--
EXPEDITED_DATA_RECEIVED	112	--

B = base, nnn = option number, [nn] = note number on pages A-53 through A-57

Product Base and Option Sets

Table A-3 (Page 6 of 6). Support for Mapped Conversation Verbs and Parameters		
Verb and Parameter	Local Support	Remote Support
MC_SEND_EXPEDITED_DATA	112	112
DATA	112	--
LENGTH	112	--
RESOURCE	112	--
WAIT_OBJECT		
(BLOCKING)	112	--
(VALUE(variable))	113 [10]	--
RETURN_CODE	112	--
REQUEST_TO_SEND_RECEIVED	112 [9]	--
EXPEDITED_DATA_RECEIVED	112	--
MC_TEST	103 or 245 [11]	--
RESOURCE	103 or 245	--
TEST		
(POSTED)	103	--
(REQUEST_TO_SEND_RECEIVED)	245	--
RETURN_CODE	103 or 245	--
B = base, nnn = option number, [nn] = note number on pages A-53 through A-57		

Type-Independent Conversation Verbs and Parameters

Table A-4 (Page 1 of 2). Support for Type-Independent Conversation Verbs and Parameters

Verb and Parameter	Local Support	Remote Support
BACKOUT	108	108
WAIT_OBJECT		
(BLOCKING)	108	--
(VALUE(variable))	108 and 113 [10]	--
RETURN_CODE	108	--
GET_TP_PROPERTIES	B	--
RETURN_CODE	B	--
OWN_TP_INSTANCE	108 and 109	--
OWN_TP_NAME	108 and 109	--
OWN_FULLY_QUALIFIED_LU_NAME	B	--
SECURITY_USER_ID	108 and 212	--
SECURITY_PROFILE	108 and 215	--
LUW_IDENTIFIER	108 and 251	--
PROTECTED_LUW_IDENTIFIER	108 and 251	--
GET_TYPE	110	--
RESOURCE	110	--
RETURN_CODE	110	--
TYPE	110 and 112	--
SET_SYNCPT_OPTIONS	108	--
OK_TO_LEAVE_OUT		
(NO)	108 and 250	--
(YES)	108 and 250	--
SELECT_LAST_AGENT		
(NO)	108 and 250	--
(YES)	108 and 250	--
VOTE_READ_ONLY_PERMITTED		
(NO)	108 and 249	--
(YES)	108 and 249	--
WAIT_FOR_OUTCOME		
(YES)	108	--
(NO)	108	--
ACTION_IF_PROBLEMS		
(BACKOUT)	108	--
(COMMIT)	108	--
RETURN_CODE	108	--

B = base, nnn = option number, [nn] = note number on pages A-53 through A-57

Product Base and Option Sets

Table A-4 (Page 2 of 2). Support for Type-Independent Conversation Verbs and Parameters		
Verb and Parameter	Local Support	Remote Support
SYNCPT	108	108
WAIT_OBJECT		
(BLOCKING)	108	--
(VALUE(variable))	108 and 113 [10]	--
RETURN_CODE	108	--
WAIT	104 [11]	--
RESOURCE_LIST	104	--
RETURN_CODE	104	--
RESOURCE_POSTED	104	--
WAIT_FOR_COMPLETION	113	--
WAIT_OBJECT_LIST	113	--
TIMEOUT	113	--
RETURN_CODE	113	--
STATUS_LIST	113	--
B = base, nnn = option number, [nn] = note number on pages A-53 through A-57		

Basic Conversation Verbs and Parameters

Table A-5 (Page 1 of 6). Support for Basic Conversation Verbs and Parameters		
Verb and Parameter	*Local Support	Remote Support
ALLOCATE	B	B [1]
LU_NAME(variable)	B or 204	B
MODE_NAME	B	B
TPN	B	B
TYPE		
(BASIC_CONVERSATION)	B	B
(MAPPED_CONVERSATION)	291	B
(FULL_DUPLEX_BASIC_CONVERSATION)	112	112
(FULL_DUPLEX_MAPPED_CONVERSATION)	112	112
RETURN_CONTROL		
(WHEN_SESSION_ALLOCATED)	B [2]	--
(IMMEDIATE)	203	--
(WHEN_CONWINNER_ALLOCATED)	201	--
(WHEN_CONVERSATION_GROUP_ALLOCATED)	401	--
(WHEN_SESSION_FREE)	205	--
CONVERSATION_GROUP_ID	401	--
SYNC_LEVEL		
(NONE)	B	B
(CONFIRM)	B [12]	B [12]
(SYNCPT)	108	108
SECURITY		
(NONE)	B	B
(SAME)	212 or 217 [3]	212, 214, 215, 216 or 217
(PGM(USER_ID(variable)	213	212, 214 or 216
PASSWORD(variable)	213	212
PROFILE(variable)))	218	215 or 217
PIP		
(NO)	B	B
(YES(variable))	241 or 242 [4]	241 or 242 [4]
WAIT_OBJECT		
(BLOCKING)	B	--
(VALUE(variable))	113 [10]	--
RETURN_CODE	B	--
RESOURCE	B	--
B = base, nnn = option number, [nn] = note number on pages A-53 through A-57		

Product Base and Option Sets

Table A-5 (Page 2 of 6). Support for Basic Conversation Verbs and Parameters

Verb and Parameter	Local Support	Remote Support
CONFIRM	B [11]	B [11]
RESOURCE	B	--
WAIT_OBJECT		
(BLOCKING)	B	--
(VALUE(variable))	113 [10]	--
RETURN_CODE	B	--
REQUEST_TO_SEND_RECEIVED	B	--
EXPEDITED_DATA_RECEIVED	112	--
CONFIRMED	B [11]	B [11]
RESOURCE	B	--
WAIT_OBJECT		
(BLOCKING)	B	--
(VALUE(variable))	113 [10]	--
RETURN_CODE	B	--
DEALLOCATE	B	B
RESOURCE	B	--
TYPE		
(SYNC_LEVEL)	B	B
(FLUSH)	B	B
(CONFIRM)	B [12]	B [12]
(ABEND_PROG)	B	B
(ABEND_SVC)	291 [18]	B
(ABEND_TIMER)	291 [18]	B
(LOCAL)	B [12]	--
LOG_DATA		
(NO)	B	B
(YES(variable))	290	B [19]
WAIT_OBJECT		
(BLOCKING)	B	--
(VALUE(variable))	113 [10]	--
RETURN_CODE	B	--
EXPEDITED_DATA_RECEIVED	112	--
FLUSH	101	B
RESOURCE	101	--
WAIT_OBJECT		
(BLOCKING)	101	--
(VALUE(variable))	113 [10]	--
RETURN_CODE	101	--

B = base, nnn = option number, [nn] = note number on pages A-53 through A-57

Table A-5 (Page 3 of 6). Support for Basic Conversation Verbs and Parameters

Verb and Parameter	Local Support	Remote Support
GET_ATTRIBUTES	B	--
RESOURCE	B	--
RETURN_CODE	B	--
PARTNER_LU_NAME	B	--
PARTNER_FULLY_QUALIFIED_LU_NAME	B	--
MODE_NAME	B	--
SYNC_LEVEL	B	--
CONVERSATION_STATE	108	--
CONVERSATION_CORRELATOR	251	--
SESSION_ID	251	--
CONVERSATION_GROUP_ID	401	--
POST_ON_RECEIPT	103 or 104 [5,11]	--
RESOURCE	103 or 104	--
LENGTH	103 or 104	--
FILL		
(LL)	103 or 104 [20]	--
(BUFFER)	103 or 104 [20]	--
RETURN_CODE	103 or 104	--
PREPARE_FOR_SYNCPT	111 [11]	--
RESOURCE	111	--
WAIT_OBJECT		
(BLOCKING)	111	--
(VALUE(variable))	113 [10]	--
RETURN_CODE	111	--
PREPARE_TO_RECEIVE	105 [11]	B [11]
RESOURCE	105	--
TYPE		
(SYNC_LEVEL)	105	B
(FLUSH)	105	B
(CONFIRM)	105	B
LOCKS		
(SHORT)	105	B
(LONG)	105 and 244	B
WAIT_OBJECT		
(BLOCKING)	105	--
(VALUE(variable))	113 [10]	--
RETURN_CODE	105	--
B = base, nnn = option number, [nn] = note number on pages A-53 through A-57		

Product Base and Option Sets

Table A-5 (Page 4 of 6). Support for Basic Conversation Verbs and Parameters

Verb and Parameter	Local Support	Remote Support
RECEIVE_AND_WAIT	B [6]	--
RESOURCE	B	--
FILL		
(LL)	B [20]	--
(BUFFER)	B [20]	--
WAIT_OBJECT		
(BLOCKING)	B	--
(VALUE(variable))	113 [10]	--
LENGTH	B	--
RETURN_CODE	B	--
REQUEST_TO_SEND_RECEIVED	B [9]	--
EXPEDITED_DATA_RECEIVED	112	--
DATA	B	--
WHAT_RECEIVED	B [7]	--
RECEIVE_EXPEDITED_DATA	112	112
RESOURCE	112	--
RETURN_CONTROL		
(WHEN_EXPEDITED_DATA_RECEIVED)	112	--
(IMMEDIATE)	112	--
WAIT_OBJECT		
(BLOCKING)	112	--
(VALUE(variable))	113 [10]	--
LENGTH	112	--
DATA	112	--
RETURN_CODE	112	--
REQUEST_TO_SEND_RECEIVED	B [9]	--
EXPEDITED_DATA_RECEIVED	112	--
RECEIVE_IMMEDIATE	106	--
RESOURCE	106	--
FILL		
(LL)	106 [20]	--
(BUFFER)	106 [20]	--
WAIT_OBJECT		
(BLOCKING)	106	--
(VALUE(variable))	113 [10]	--
LENGTH	106	--
RETURN_CODE	106	--
REQUEST_TO_SEND_RECEIVED	106 [9]	--
EXPEDITED_DATA_RECEIVED	112	--
DATA	106	--
WHAT_RECEIVED	106 [7]	--

B = base, nnn = option number, [nn] = note number on pages A-53 through A-57

Table A-5 (Page 5 of 6). Support for Basic Conversation Verbs and Parameters

Verb and Parameter	Local Support	Remote Support
REQUEST_TO_SEND	B [8,11]	B [11]
RESOURCE	B	--
WAIT_OBJECT		
(BLOCKING)	B	--
(VALUE(variable))	113 [10]	--
RETURN_CODE	B	--
SEND_DATA	B	B
RESOURCE	B	--
DATA	B	B
LENGTH	B	B
ENCRYPT		
(NO)	B	B
(YES)	611 or 617	B
WAIT_OBJECT		
(BLOCKING)	B	--
(VALUE(variable))	113 [10]	--
RETURN_CODE	B	--
REQUEST_TO_SEND_RECEIVED	B [9]	--
EXPEDITED_DATA_RECEIVED	112	--
SEND_ERROR	B	B
RESOURCE	B	--
TYPE		
(PROG)	B	B
(SVC)	291 [18]	B
LOG_DATA		
(NO)	B	B
(YES(variable))	290	B [19]
WAIT_OBJECT		
(BLOCKING)	B	--
(VALUE(variable))	113 [10]	--
RETURN_CODE	B	--
REQUEST_TO_SEND_RECEIVED	B [9]	--
EXPEDITED_DATA_RECEIVED	112	--
SEND_EXPEDITED_DATA	112	112
DATA	112	--
LENGTH	112	--
RESOURCE	112	--
WAIT_OBJECT		
(BLOCKING)	112	--
(VALUE(variable))	113 [10]	--
RETURN_CODE	112	--
REQUEST_TO_SEND_RECEIVED	B [9]	--
EXPEDITED_DATA_RECEIVED	112	--

B = base, nnn = option number, [nn] = note number on pages A-53 through A-57

Product Base and Option Sets

Table A-5 (Page 6 of 6). Support for Basic Conversation Verbs and Parameters		
Verb and Parameter	Local Support	Remote Support
TEST	103 or 245 [11]	--
RESOURCE	103 or 245	--
TEST		
(POSTED)	103	--
(REQUEST_TO_SEND_RECEIVED)	245	--
RETURN_CODE	103 or 245	--
B = base, nnn = option number, [nn] = note number on pages A-53 through A-57		

Conversation Return Codes and What-Received Indications

Table A-6 (Page 1 of 2). Support for Conversation Return Codes

Return Code and Subcode	Local Support
ALL_UNKNOWN_WAIT_OBJECTS	113
ALLOCATION_ERROR	B
ALLOCATION_FAILURE_NO_RETRY	B
ALLOCATION_FAILURE_RETRY	B
CONVERSATION_TYPE_MISMATCH	B
FULL_DUPLEX_NOT_SUPPORTED_BY_LU	112
PIP_NOT_SPECIFIED_CORRECTLY	B
SECURITY_NOT_VALID	B
SYNC_LEVEL_NOT_SUPPORTED_BY_LU	108
SYNC_LEVEL_NOT_SUPPORTED_BY_PGM	SEND: 108 RCV: B
TPN_NOT_RECOGNIZED	B
TP_NOT_AVAIL_NO_RETRY	B
TP_NOT_AVAIL_RETRY	B
BACKED_OUT	108
ALL_AGREED	108
LUW_OUTCOME_PENDING	108
LUW_OUTCOME_MIXED	108
CONVERSATION_ENDED	112
DEALLOCATE_ABEND	B
DEALLOCATE_ABEND_PROG	B
DEALLOCATE_ABEND_SVC	B
DEALLOCATE_ABEND_TIMER	B
DEALLOCATE_NORMAL	B
ENCRYPTION_NOT_SUPPORTED	617
ERROR_INDICATION	112
ALLOCATION_ERROR	112
DEALLOCATE_ABEND	112
DEALLOCATE_ABEND_PROG	112
DEALLOCATE_ABEND_SVC	112
DEALLOCATE_ABEND_TIMER	112
RESOURCE_FAILURE_NO_RETRY	112
RESOURCE_FAILURE_RETRY	112
UNKNOWN	112
EXPEDITED_DATA_NOT_SUPPORTED_BY_LU	112
INSUFFICIENT_BUFFER_SIZE_PROVIDED	112
INVALID_TIMEOUT_VALUE	113
MAP_EXECUTION_FAILURE	246
MAP_NOT_FOUND	246
MAPPING_NOT_SUPPORTED	246
B = base, nnn = option number, [nn] = note number (pp. A-53 - A-57)	

Product Base and Option Sets

Table A-6 (Page 2 of 2). Support for Conversation Return Codes	
Return Code and Subcode	Local Support
OK	B
DATA *	103 or 104
NOT_DATA	103 or 104
ALL_AGREED	108
VOTED_READ_ONLY	249
LUW_OUTCOME_PENDING	108
LUW_OUTCOME_MIXED	108
OPERATION_INCOMPLETE	113
OPERATION_NOT_ACCEPTED	113
PARAMETER_ERROR	B
POSTING_NOT_ACTIVE	103 or 104
PROG_ERROR_PURGING	B [13]
PROG_ERROR_NO_TRUNC	B
PROGRAM_PARAMETER_CHECK	B
PROGRAM_STATE_CHECK	B
RESOURCE_FAILURE_NO_RETRY	B
RESOURCE_FAILURE_RETRY	B
SVC_ERROR_NO_TRUNC	B
SVC_ERROR_PURGING	B [13]
SVC_ERROR_TRUNC	B
TIMER_EXPIRED	113
UNSUCCESSFUL	103, 106, 112 203 or 245
USER_CONTROL_DATA_NOT_SUPPORTED	247
B = base, nnn = option number, [nn] = note number (pp. A-53 - A-57)	

Table A-7. Support for Conversation What-Received Indications

What-Received Indication	Local Support
CONFIRM	B [13]
CONFIRM_DEALLOCATE	B [13]
CONFIRM_SEND	B [13]
DATA	B
DATA_COMPLETE	B
DATA_INCOMPLETE	B
DATA_TRUNCATED	B [17]
LL_TRUNCATED	B [21]
SEND	B [13]
TAKE_SYNCPT	108
TAKE_SYNCPT_DEALLOCATE	108
TAKE_SYNCPT_SEND	108
USER_CONTROL_DATA_COMPLETE	247
USER_CONTROL_DATA_INCOMPLETE	247
USER_CONTROL_DATA_TRUNCATED	247 [17]
B = base, nnn = option number, [nn] = note number (pp. A-53 - A-57)	

Support for Control-Operator Verbs

Control-Operator Verbs and Parameters for CNOS

Table A-8 (Page 1 of 2). Support for Control-Operator Verbs and Parameters for CNOS

Verb and Parameter	Local Support	Remote Support
CHANGE_SESSION_LIMIT	501	B
LU_NAME	501	B
MODE_NAME	501	B
LU_MODE_SESSION_LIMIT	501	B
MIN_CONWINNERS_SOURCE	501	B
MIN_CONWINNERS_TARGET	601	B
RESPONSIBLE(SOURCE)	501	B
RESPONSIBLE(TARGET)	602	B [23]
RETURN_CODE	501	--
INITIALIZE_SESSION_LIMIT	B	B
LU_NAME	B	B
MODE_NAME(variable)	B	B
MODE_NAME('SNASVCMG')	B	B
MODE_NAME('CPSVCMG')	616	--
LU_MODE_SESSION_LIMIT	B	B
MIN_CONWINNERS_SOURCE	B	B
MIN_CONWINNERS_TARGET	601	B
RETURN_CODE	B	--
PROCESS_SESSION_LIMIT	B	B
RESOURCE	B	--
RETURN_CODE	B	--
LU_NAME	B	--
MODE_NAME	B	--
B = base, nnn = option number, [nn] = note number on pages A-53 through A-57		

Table A-8 (Page 2 of 2). Support for Control-Operator Verbs and Parameters for CNOS		
Verb and Parameter	Local Support	Remote Support
RESET_SESSION_LIMIT	B	B
LU_NAME	B	B
MODE_NAME(ALL)	B	B
MODE_NAME(ONE(variable))	B	B
MODE_NAME(ONE('SNASVCMG'))	B	B
MODE_NAME('CPSVCMG')	616	--
RESPONSIBLE(SOURCE)	B	B
RESPONSIBLE(TARGET)	602	B [23]
DRAIN_SOURCE(NO)	B [22]	B
DRAIN_SOURCE(YES)	B [22]	B
DRAIN_TARGET(NO)	603	B
DRAIN_TARGET(YES)	B	B [24]
FORCE(NO)	B	B
FORCE(YES)	604	B
RETURN_CODE	B	--
B = base, nnn = option number, [nn] = note number on pages A-53 through A-57		

Control-Operator Verbs and Parameters for Session Control

Table A-9. Support for Control-Operator Verbs and Parameters for Session Control		
Verb and Parameter	Local Support	Remote Support
ACTIVATE_SESSION	502	B
LU_NAME	502	B
MODE_NAME(variable)	502	B
MODE_NAME('SNASVCMG')	502	B
MODE_NAME('CPSVCMG')	502	B
RETURN_CODE	502	--
DEACTIVATE_CONVERSATION_GROUP	401	--
CONVERSATION_GROUP_ID	401	--
TYPE(CLEANUP)	401	--
TYPE(NORMAL)	401	--
SENSE_CODE(NO)	401	--
SENSE_CODE(YES(variable))	401	--
RETURN_CONTROL(IMMEDIATE)	401 and 504	--
RETURN_CONTROL(DELAYED)	401, 108, and 504	--
RETURN_CODE	401	--
DEACTIVATE_SESSION	504	B
SESSION_ID	504	--
TYPE(CLEANUP)	504	B
TYPE(NORMAL)	504	B
RETURN_CONTROL(IMMEDIATE)	504	--
RETURN_CONTROL(DELAYED)	504 and 108	--
RETURN_CODE	504	--
B = base, nnn = option number, [nn] = note number on pages A-53 through A-57		

Control-Operator Verbs and Parameters for LU Definition

Table A-10 (Page 1 of 6). Support for Control-Operator Verbs and Parameters for LU Definition		
Verb and Parameter	Local Support	Remote Support
DEFINE_LOCAL_LU	505	--
FULLY_QUALIFIED_LU_NAME	505	--
LU_SESSION_LIMIT(NONE)	505	--
LU_SESSION_LIMIT(VALUE(variable))	505 and 605	--
SECURITY(ADD(USER_ID(variable) PASSWORD(variable) PROFILE(variable)))	505 and 212	--
SECURITY(DELETE(USER_ID(variable) PROFILE(variable)))	505 and 212	--
MAP_NAME(ADD(variable))	505 and 215	--
MAP_NAME(DELETE(variable))	505 and 215	--
LENGTH_CHECKED_COMPRESSION	505 and 246	--
INBOUND_COMPRESSION_LEVEL	505 and 246	--
OUTBOUND_COMPRESSION_LEVEL	618	--
RLE_BEFORE_LZ	618	--
RETURN_CODE	618	--
DEFINE_MODE	505	--
FULLY_QUALIFIED_LU_NAME	505 [25]	--
MODE_NAME	505	--
SEND_PACING_WINDOW	505	--
RECEIVE_PACING_WINDOW	505	--
SEND_MAX_RU_SIZE_LOWER_BOUND(256)	505	--
SEND_MAX_RU_SIZE_LOWER_BOUND(-256)	505 and 610	--
SEND_MAX_RU_SIZE_UPPER_BOUND(256)	505	--
SEND_MAX_RU_SIZE_UPPER_BOUND(-256)	505 and 610	--
RECEIVE_MAX_RU_SIZE_LOWER_BOUND(256)	505	--
RECEIVE_MAX_RU_SIZE_LOWER_BOUND(-256)	505 and 610	--
RECEIVE_MAX_RU_SIZE_UPPER_BOUND(256)	505	--
RECEIVE_MAX_RU_SIZE_UPPER_BOUND(-256)	505 and 610	--
B = base, nnn = option number, [nn] = note number on pages A-53 through A-57		

Product Base and Option Sets

Table A-10 (Page 2 of 6). Support for Control-Operator Verbs and Parameters for LU Definition		
Verb and Parameter	Local Support	Remote Support
SINGLE_SESSION_REINITIATION(OPERATOR)	505	--
SINGLE_SESSION_REINITIATION(PLU)	505 and 608	--
SINGLE_SESSION_REINITIATION(SLU)	505 and 608	--
SINGLE_SESSION_REINITIATION(PLU_OR_SLU)	505 and 608	--
SESSION_LEVEL_CRYPTOGRAPHY(NO)	505	--
SESSION_LEVEL_CRYPTOGRAPHY(MANDATORY)	505 and 611	--
CONWINNER_AUTO_ACTIVATE_LIMIT	505 and 612	--
LENGTH_CHECKED_COMPRESSION	618	--
INBOUND_COMPRESSION_LEVEL	618	--
OUTBOUND_COMPRESSION_LEVEL	618	--
RLE_BEFORE_LZ	618	--
RETURN_CODE	505	--
DEFINE_REMOTE_LU	505	--
FULLY_QUALIFIED_LU_NAME	505	--
LOCALLY_KNOWN_LU_NAME(NONE)	505	--
LOCALLY_KNOWN_LU_NAME(NAME(variable))	505 and 606	--
UNINTERPRETED_LU_NAME(NONE)	505	--
UNINTERPRETED_LU_NAME(NAME(variable))	505 and 607	--
INITIATE_TYPE(INITIATE_ONLY)	505	--
INITIATE_TYPE(INITIATE_OR_QUEUE)	505	--
PARALLEL_SESSION_SUPPORT(YES)	505	--
PARALLEL_SESSION_SUPPORT(NO)	505	--
CNOS_SUPPORT(YES)	505	--
CNOS_SUPPORT(NO)	505	--
LU_LU_PASSWORD(NONE)	505	--
LU_LU_PASSWORD(VALUE(variable))	505 and 211	--
SECURITY_ACCEPTANCE(NONE)	505	--
SECURITY_ACCEPTANCE(CONVERSATION)	505 and 212	--
SECURITY_ACCEPTANCE(ALREADY_VERIFIED)	505 and 212	--
SECURITY_ACCEPTANCE(PERSISTENT_VERIFICATION)	505 and 220	--
SECURITY_ACCEPTANCE(AV_AND_PV)	505, 212, and 220	--
RETURN_CODE	505	--
B = base, nnn = option number, [nn] = note number on pages A-53 through A-57		

Table A-10 (Page 3 of 6). Support for Control-Operator Verbs and Parameters for LU Definition		
Verb and Parameter	Local Support	Remote Support
DEFINE_TP	505	--
TP_NAME	505	--
STATUS(ENABLED)	505	--
STATUS(TEMP_DISABLED)	505	--
STATUS(PERM_DISABLED)	505	--
CONVERSATION_TYPE(MAPPED)	505	--
CONVERSATION_TYPE(BASIC)	505	--
CONVERSATION_TYPE(FULL_DUPLEX_MAPPED)	505 and 112	--
CONVERSATION_TYPE(FULL_DUPLEX_BASIC)	505 and 112	--
SYNC_LEVEL(NONE)	505	--
SYNC_LEVEL(CONFIRM)	505	--
SYNC_LEVEL(SYNCPT)	505 and 108	--
SECURITY_REQUIRED(NONE)	505	--
SECURITY_REQUIRED(CONVERSATION)	505 and 212	--
SECURITY_REQUIRED(ACCESS(USER_ID))	505 and 214	--
SECURITY_REQUIRED(ACCESS(PROFILE))	505 and 215	--
SECURITY_REQUIRED(ACCESS(USER_ID_PROFILE))	505 and 214, 215	--
SECURITY_REQUIRED(ACCESS(USER_ID_REMOTE_LU))	505 and 214, 216	--
SECURITY_REQUIRED(ACCESS(USER_ID_PROFILE_REMOTE_LU))	505 and 214, 215 216	--
SECURITY_ACCESS(ADD(USER_ID(variable) PROFILE(variable) REMOTE_LU(variable)))	505 and 214	--
	505 and 215	--
	505 and 216	--
SECURITY_ACCESS(DELETE(USER_ID(variable) PROFILE(variable) REMOTE_LU(variable)))	505 and 214	--
	505 and 215	--
	505 and 216	--
PIP(NO)	505	--
PIP(YES(variable))	505 and 241 or 242	--
PIP(NO_LU_VERIFICATION)	505	--
DATA_MAPPING(NO)	505	--
DATA_MAPPING(YES)	505 and 246	--
B = base, nnn = option number, [nn] = note number on pages A-53 through A-57		

Product Base and Option Sets

Table A-10 (Page 4 of 6). Support for Control-Operator Verbs and Parameters for LU Definition

Verb and Parameter	Local Support	Remote Support
PRIVILEGE(NONE)	505	--
PRIVILEGE(CNOS)	505	--
PRIVILEGE(SESSION_CONTROL)	505 and 502 or 504	--
PRIVILEGE(DEFINE)	505	--
PRIVILEGE(DISPLAY)	505	--
PRIVILEGE(ALLOCATE_SERVICE_TP)	505	--
INSTANCE_LIMIT	505 and 401	--
RETURN_CODE	505	--
DELETE	505	--
LOCAL_LU_NAME	505	--
REMOTE_LU_NAME	505	--
MODE_NAME	505	--
TP_NAME	505	--
RETURN_CODE	505	--
DISPLAY_LOCAL_LU	505	--
FULLY_QUALIFIED_LU_NAME	505	--
RETURN_CODE	505	--
LU_SESSION_LIMIT	505	--
LU_SESSION_COUNT	505	--
SECURITY	505 and 212	--
MAP_NAMES	505	--
REMOTE_LU_NAMES	505	--
TP_NAMES	505	--
LENGTH_CHECKED_COMPRESSION	618	--
INBOUND_COMPRESSION_LEVEL	618	--
OUTBOUND_COMPRESSION_LEVEL	618	--
RLE_BEFORE_LZ	618	--
B = base, nnn = option number, [nn] = note number on pages A-53 through A-57		

Table A-10 (Page 5 of 6). Support for Control-Operator Verbs and Parameters for LU Definition

Verb and Parameter	Local Support	Remote Support
DISPLAY_MODE	505	--
FULLY_QUALIFIED_LU_NAME	505	--
MODE_NAME	505	--
RETURN_CODE	505	--
SEND_PACING_WINDOW	505	--
RECEIVE_PACING_WINDOW	505	--
SEND_MAX_RU_SIZE_LOWER_BOUND	505 and 610	--
SEND_MAX_RU_SIZE_UPPER_BOUND	505 and 610	--
RECEIVE_MAX_RU_SIZE_LOWER_BOUND	505 and 610	--
RECEIVE_MAX_RU_SIZE_UPPER_BOUND	505 and 610	--
SINGLE_SESSION_REINITIATION	505 and 608	--
SESSION_LEVEL_CRYPTOGRAPHY	505 and 611	--
CONWINNER_AUTO_ACTIVATE_LIMIT	505 and 612	--
LOCAL_MAX_SESSION_LIMIT	505 and 613	--
LU_MODE_SESSION_LIMIT	505	--
MIN_CONWINNERS	505	--
MIN_CONLOSERS	505	--
TERMINATION_COUNT	505	--
DRAIN_LOCAL_LU	505	--
DRAIN_REMOTE_LU	505	--
LU_MODE_SESSION_COUNT	505	--
CONWINNERS_SESSION_COUNT	505	--
CONLOSERS_SESSION_COUNT	505	--
SESSION_IDS	505	--
CONVERSATION_GROUP_IDS	505 and 401	--
PREFERRED_RECEIVE_RU_SIZE	505	--
PREFERRED_SEND_RU_SIZE	505	--
SESSION_DEACTIVATED_TP_NAME	505 and 401	--
LENGTH_CHECKED_COMPRESSION	618	--
INBOUND_COMPRESSION_LEVEL	618	--
OUTBOUND_COMPRESSION_LEVEL	618	--
RLE_BEFORE_LZ	618	--
DISPLAY_REMOTE_LU	505	--
FULLY_QUALIFIED_LU_NAME	505	--
RETURN_CODE	505	--
LOCALLY_KNOWN_LU_NAME	505 and 606	--
UNINTERPRETED_LU_NAME	505 and 607	--
INITIATE_TYPE	505	--
PARALLEL_SESSION_SUPPORT	505	--
CNOS_SUPPORT	505	--
SECURITY_ACCEPTANCE_LOCAL_LU	505 and 212	--
SECURITY_ACCEPTANCE_REMOTE_LU	505 and 212	--
MODE_NAMES	505	--
B = base, nnn = option number, [nn] = note number on pages A-53 through A-57		

Product Base and Option Sets

Table A-10 (Page 6 of 6). Support for Control-Operator Verbs and Parameters for LU Definition

Verb and Parameter	Local Support	Remote Support
DISPLAY_TP	505	--
TP_NAME	505	--
RETURN_CODE	505	--
STATUS	505	--
CONVERSATION_TYPE	505 and 112	--
SYNC_LEVEL	505 and 108	--
SECURITY_REQUIRED	505 and 212 or 214 or 215 215 or 216	--
SECURITY_ACCESS	505 and 214, 215, or 216	--
PIP	505 and 241 or 242	--
DATA_MAPPING	505 and 246	--
PRIVILEGE	505	--
INSTANCE_LIMIT	505	--
INSTANCE_COUNT	505	--
B = base, nnn = option number, [nn] = note number on pages A-53 through A-57		

Control-Operator Verbs and Parameters for Miscellaneous Verbs

Table A-11. Support for Miscellaneous Control-Operator Verbs and Parameters

Verb and Parameter	Local Support	Remote Support
DISPLAY_SIGNED_ON_LIST	219 or 220	--
LIST(ALL_SIGNED_ON_LISTS)	219 or 220	--
LIST(SIGNED_ON_TO_LIST)	219	--
LIST(SIGNED_ON_FROM_LIST)	220	--
SIGNED_ON_TO_LIST	219	--
ENTRIES_ON_TO_LIST	219	--
SIGNED_ON_FROM_LIST	220	--
ENTRIES_ON_FROM_LIST	220	--
RETURN_CODE	219 or 220	--
PROCESS_SIGNOFF	219	220
RESOURCE	219	220
RETURN_CODE	219	220
SIGNOFF	219 or 220	219 or 220
LIST(ALL_SIGNED_ON_LISTS)	219 or 220	219 or 220
(SIGNED_ON_TO_LIST)	219	220
(SIGNED_ON_FROM_LISTS)	220	219
USER(ALL)	219 or 220	219 or 220
(USERID(variable))	219 or 220	219 or 220
PROFILE(ALL)	219 or 220	219 or 220
(PROFILE(variable))	219 or 220	219 or 220
LU(ALL)	219 or 220	219 or 220
(LU_NAME(variable))	219 or 220	219 or 220
ADDITIONAL_CRITERIA(NONE)	219 or 220	219 or 220
(REMOVE_INACTIVE(variables))	219 or 220	219 or 220
RETURN_CODE	219 or 220	219 or 220
B = base, nnn = option number, [nn] = note number on pages A-53 through A-57		

Control-Operator Return Codes

Table A-12. Support for Control-Operator Return Codes	
Return Code and Subcode *	Local Support
ACTIVATION_FAILURE_NO_RETRY	502
ACTIVATION_FAILURE_RETRY	502
ALLOCATION_ERROR	B
ALLOCATION_FAILURE_NO_RETRY	B
ALLOCATION_FAILURE_RETRY	B
TRANS_PGM_NOT_AVAIL_RETRY	B
COMMAND_RACE_REJECT	B
LU_MODE_SESSION_LIMIT_CLOSED	B
LU_MODE_SESSION_LIMIT_EXCEEDED	502
LU_MODE_SESSION_LIMIT_NOT_ZERO	B
LU_MODE_SESSION_LIMIT_ZERO	501
LU_SESSION_LIMIT_EXCEEDED	B
OK	B
AS_SPECIFIED	B
AS_NEGOTIATED	B
FORCED	604
PARAMETER_ERROR	B
REQUEST_EXCEEDS_MAX_ALLOWED	613
RESOURCE_FAILURE_NO_RETRY	B
UNRECOGNIZED_MODE_NAME	B
B = base, nnn = option number, [nn] = note number (pages A-53 - A-57)	

Notes on Implementation Details

The following notes pertain to the base and optional support shown in the preceding tables. These notes describe certain implementation details, which are product dependent.

Notes That Apply to Conversation Verbs (Basic or Mapped):

1. The MC_ALLOCATE and ALLOCATE verbs send an allocation request to the remote LU. The remote LU starts the transaction program named in the allocation request. As this function is described in this book, the remote LU starts a new execution instance of the named transaction program. All products support this capability. A product may, in addition, allow an already-executing instance of the named transaction program to receive an allocation request by means of a product-dependent verb. This product-dependent capability provides no means in the LU for correlating the new conversation to a previous one.
2. The RETURN_CONTROL parameter on the MC_ALLOCATE and ALLOCATE verbs specifies when local processing of the verb is to be completed, in terms of the allocation of a session for the conversation. A product may provide for the specification of additional conditions for allocating sessions, such as a variation of the argument WHEN_SESSION_ALLOCATED that omits contention-loser sessions from the selection process.
3. Products that provide local support for option set 212 (conversation-level security verification) but not option set 213 (program-supplied user ID and password) may choose to not make the SECURITY parameter on the MC_ALLOCATE and ALLOCATE verbs explicitly available to their transaction programs. Such products implicitly support both SECURITY(SAME) and SECURITY(NONE) in that they downgrade SECURITY(SAME) to SECURITY(NONE) when the remote LU does not accept conversation-level security, or the already-verified indication, from the local LU.
4. Support of the PIP parameter on the MC_ALLOCATE and ALLOCATE verbs is optional and local support is independent of remote support. A product may provide either local or remote support, or both.
5. The MC_POST_ON_RECEIPT and POST_ON_RECEIPT verbs, as described in this book, may be issued for a given conversation any number of times before posting is reset or canceled. (See the notes under the descriptions of the verbs for more details.) However, a product may, instead, permit the verbs to be issued only once for a given conversation and disallow subsequent use of the verbs on that conversation until posting is reset or canceled.
6. The MC_RECEIVE_AND_WAIT and RECEIVE_AND_WAIT verbs are used to wait until the requested amount of data or other information is available for the program to receive, receive the data or other information, and then resume program execution. Rather than resuming program execution as soon as the requested amount of data is received, the product may defer resuming program execution until it receives something other than data, such as a SEND, CONFIRM, TAKE_SYNCPT, or DEALLOCATE indication.
7. The WHAT_RECEIVED parameter on the MC_RECEIVE_AND_WAIT, MC_RECEIVE_IMMEDIATE, RECEIVE_AND_WAIT, and

RECEIVE_IMMEDIATE verbs returns to the transaction program only one indication at a time. This serialization of returning what-received indications to the program results in discrete states of the conversation for each returned indication. (See the verb descriptions in this book for the state changes that can occur.) A product may, instead, return more than one indication at a time. For example, the product may return indications for both DATA and SEND at the completion of the verbs. In this case, the state changes that occur at the completion of the verbs may differ from that described in this book. Refer to the product's publication for further details.

8. The MC_REQUEST_TO_SEND and REQUEST_TO_SEND verbs, as described in this book, may be issued only when the conversation is in send, receive, confirm, or sync point state. Issuing the verbs for a conversation in any other state is described as a state-check ABEND condition. As an alternative to the ABEND condition, a product may also permit its transaction programs to issue the verbs for conversations in send or defer state.
9. The REQUEST_TO_SEND_RECEIVED parameter is used to receive a request-to-send notification on the MC_RECEIVE_IMMEDIATE and RECEIVE_IMMEDIATE verbs when the conversation is in receive state and on the MC_RECEIVE_AND_WAIT and RECEIVE_AND_WAIT verbs when the conversation is in send or receive state. (See the notes under the descriptions of these verbs for details of when this can occur.) However, a product may defer passing the request-to-send notification to the program until the program issues an MC_SEND_DATA or SEND_DATA verb, or issues an MC_SEND_ERROR or SEND_ERROR verb when the conversation is in send state.

Also, a product that defers passing the request-to-send notification to the program may discard the notification, and not pass it to the program, if the program issues an MC_PREPARE_TO_RECEIVE or PREPARE_TO_RECEIVE verb, issues an MC_RECEIVE_AND_WAIT or RECEIVE_AND_WAIT verb when the conversation is in send state, or receives a PROG_ERROR_PURGING or SVC_ERROR_PURGING return code when the conversation is in send state.

The REQUEST_TO_SEND_RECEIVED indicator is always set to a value of NO for full-duplex conversations (option set 112).

10. Support of the WAIT_OBJECT parameter on all the verbs whose execution can be blocked is optional and is only a local function. An implementation may provide this support in the local program independent of support in the remote program.
11. The following verbs are not supported for full-duplex conversations (option set 112):
 - (MC_)CONFIRM
 - (MC_)CONFIRMED
 - (MC_)POST_ON_RECEIPT
 - (MC_)PREPARE_FOR_SYNCPT
 - (MC_)PREPARE_TO_RECEIVE
 - (MC_)RECONNECT
 - (MC_)REQUEST_TO_SEND
 - (MC_)TEST

- WAIT
12. The following verb parameters are not supported for full-duplex conversations (option set 112):
- | Conversation Verb | Parameters not supported for full-duplex conversations |
|-------------------|--|
| (MC_)ALLOCATE | SYNC_LEVEL(CONFIRM) |
| (MC_)DEALLOCATE | TYPE(CONFIRM), TYPE(LOCAL) |
13. The following return codes and what-received indications are not supported for full-duplex conversations (option set 112).
- return codes
 - PROG_ERROR_PURGING
 - SVC_ERROR_PURGING
 - what-received indications
 - CONFIRM
 - CONFIRM_DEALLOCATE
 - CONFIRM_SEND
 - SEND

Notes That Apply Only to Mapped Conversation Verbs:

14. A base local and remote support is defined for the data record length specified by the LENGTH parameter on the MC_POST_ON_RECEIPT, MC_RECEIVE_AND_WAIT, MC_RECEIVE_IMMEDIATE, and MC_SEND_DATA verbs. The base support for the data record length is 2048. All transaction programs are allowed to send and receive data records up to 2048 bytes in length. Local and remote support for data records greater than 2048 bytes in length is optional, and the maximum length is product-dependent.
15. The MAP_NAME parameter on the MC_SEND_DATA verb is used to specify data mapping. MAP_NAME(NO) yields a null value for the map name, which suppresses data mapping, whereas MAP_NAME(YES(variable)) specifies a non-null map name, which invokes data mapping. Products that support option set 246 (data mapping) provide local and remote support for both MAP_NAME(NO) and MAP_NAME(YES(variable)). However, a product may provide local support only for MAP_NAME(YES(variable)) on MC_SEND_DATAs issued on conversations that use data mapping.
16. The USER_CONTROL_DATA(YES) parameter on the MC_SEND_DATA verb specifies that the data record contains user control data. For example, transaction programs written for a product that implements LU 6.1 make use of the USER_CONTROL_DATA field for sending FM header data. A product that implements LU 6.2 may provide local and remote support for this parameter, either because it processes LU 6.1 transaction programs or because it processes LU 6.2 transaction programs that connect to LU 6.1 transaction programs.
17. The what-received indications, DATA_TRUNCATED and USER_CONTROL_DATA_TRUNCATED, inform the program that it received only part of the data record and the LU has discarded the remaining part. A product may, instead, retain the remaining data and indicate

DATA_INCOMPLETE or USER_CONTROL_DATA_INCOMPLETE.

Alternatively, the product may support both capabilities and allow the program to select whether the LU is to discard or retain remaining data.

Notes That Apply Only to Basic Conversation Verbs:

18. The TYPE(ABEND_SVC) and TYPE(ABEND_TIMER) parameters on the DEALLOCATE verb and the TYPE(SVC) parameter on the SEND_ERROR verb are used to indicate errors that the mapped conversation LU services component detects. A product that does not support option set 291 (mapped conversation LU services component) may, nevertheless, support these parameters at its basic conversation protocol boundary.

The TYPE(ABEND_PROG) parameter on the DEALLOCATE verb is typically used by transaction programs while the TYPE(ABEND_SVC) parameter on the DEALLOCATE verb is typically used by the LU. This allows for distinguishing between transaction program errors and system errors.

19. The LOG_DATA parameter on the DEALLOCATE and SEND_ERROR verbs is used to record product-unique error information in the system error logs of the local and remote LUs. The capability to receive the log data is part of the base remote support. However, a product that does not provide local support for option set 290 (logging of data in a system log) may discard the received log data rather than process it.

20. The FILL parameter on the POST_ON_RECEIPT, RECEIVE_AND_WAIT, and RECEIVE_IMMEDIATE verbs has two arguments: LL and BUFFER. A product may support only one of the arguments, or it may support both of them. All products implemented at the level of SNA defined in the fourth or later edition of this book and all products supporting CPI for Communications always support both FILL parameters in their APIs.

The arguments are described in this book as being independent of each other; that is, the specification of either one does not depend on the past use of the parameter, and has no bearing on its subsequent use. A product supporting both arguments may treat them as described, or it may treat them in a dependent manner, allowing the program to specify only one or the other for certain sequences of the verbs and indicating an error if this restriction is violated.

21. The what-received indication, LL_TRUNCATED, informs the program that the LU received only the first byte of the LL field of a logical record, because it was truncated. The truncated LL field is discarded by the receiving LU rather than being passed to the receiving program. A product may, instead, pass the truncated LL field to the program and indicate DATA_INCOMPLETE rather than LL_TRUNCATED.

Notes That Apply to Control-Operator Verbs:

22. A product may provide local (source LU) support for only one of the arguments, NO or YES, of the DRAIN_SOURCE parameter on the RESET_SESSION_LIMIT verb, or it may support both arguments. However, all products provide remote (target LU) support for both arguments.
23. Remote support for the RESPONSIBLE(TARGET) parameter of the CHANGE_SESSION_LIMIT and RESET_SESSION_LIMIT verbs is part of

the base set of functions. However, a product may provide remote support for this parameter by always negotiating the TARGET argument to SOURCE during its remote processing, as a target LU, of CHANGE_SESSION_LIMIT and RESET_SESSION_LIMIT.

24. Remote support for the DRAIN_TARGET(YES) parameter of the the RESET_SESSION_LIMIT verb is part of the base set of functions. However, a product may provide remote support for this parameter by always negotiating the YES argument to NO during its remote processing, as a target LU, of RESET_SESSION_LIMIT.
25. A product may allow the control operator or transaction program to specify certain parameters of the DEFINE_MODE verb by means of the DEFINE_LOCAL_LU or DEFINE_REMOTE_LU verb, instead. These parameters are:

SINGLE_SESSION_REINITIATION
SESSION_LEVEL_CRYPTOGRAPHY

When these parameters are specified by means of DEFINE_LOCAL_LU, the local LU's corresponding operating parameters are constant across all mode names for all remote LUs. Similarly, when these parameters are specified by means of DEFINE_REMOTE_LU, the local LU's corresponding operating parameters are constant across all mode names for a given remote LU, but they may differ for each remote LU.

Product Base and Option Sets

Appendix B. Examples Using Basic Conversation Verbs

This appendix contains examples of the use of some of the basic conversation verbs. Each example shows two transaction programs, TP(a) and TP(b), connected by a conversation. The letters a and b represent each program's name.

Each example concentrates on the use of a small number of verbs and how the verbs issued by one program relate to the verbs issued by the other program. When a verb causes the LU to send information to the other program, the resulting flow is shown as an arrow (—→). Some verbs cause the LU to suspend the program's processing until the LU completes execution of the verb; a vertical dotted line under the verb indicates the suspension of program processing.

Some parameters are shown with the verbs. The parameters shown are those that are significant to the example. Supplied parameters are shown as "parameter-name(supplied-value)," and returned parameters are shown as "parameter-name = returned-value." Parameters not significant to the example are not shown.

On the page facing each example are notes that explain what the example is illustrating. The notes are numbered. The part of the example to which the note applies is keyed with the same number, shown within brackets. For instance, the part of the example in Figure B-1 on page B-4 that is keyed by "[1]" is explained by note 1 on the facing page.

The examples contain a few comments, which are shown within brackets. For example, the comment, "[TP(a) running]," in Figure B-1 on page B-4 means the program is already processing at the point the example begins.

The examples are organized, front to back, in somewhat chronological order in relation to the allocation and use of a conversation; they are intended to be read in this order. In particular, the early examples begin with the allocation of a conversation between two programs. The later examples omit the allocation process and, instead, begin with both programs already in conversation with each other. Additionally, where the same verb appears in more than one example, the notes associated with the verb in the early examples may provide more detailed information than the notes given in the later examples.

The examples are simplified with the following assumptions:

- The sending LU buffers (retains for later transmission) allocation requests.
- The sending LU sends error notifications immediately.
- In all examples except Figure B-8, the LU's send buffer is large enough to hold all the information its TP sends, and transmits the information only when its TP causes it to do so, such as by issuing the DEALLOCATE verb.
- The TP sending data sends one logical record with each SEND_DATA verb.
- The TP receiving data receives one logical record with each RECEIVE_AND_WAIT verb.

Examples Using Basic Conversation Verbs

These simplifications reflect implementation, installation, and application-program choices permitted in the architecture. See Part 2, "Conversation Verbs" for more details.

Examples Using Basic Conversation Verbs

Examples Using Basic Conversation Verbs

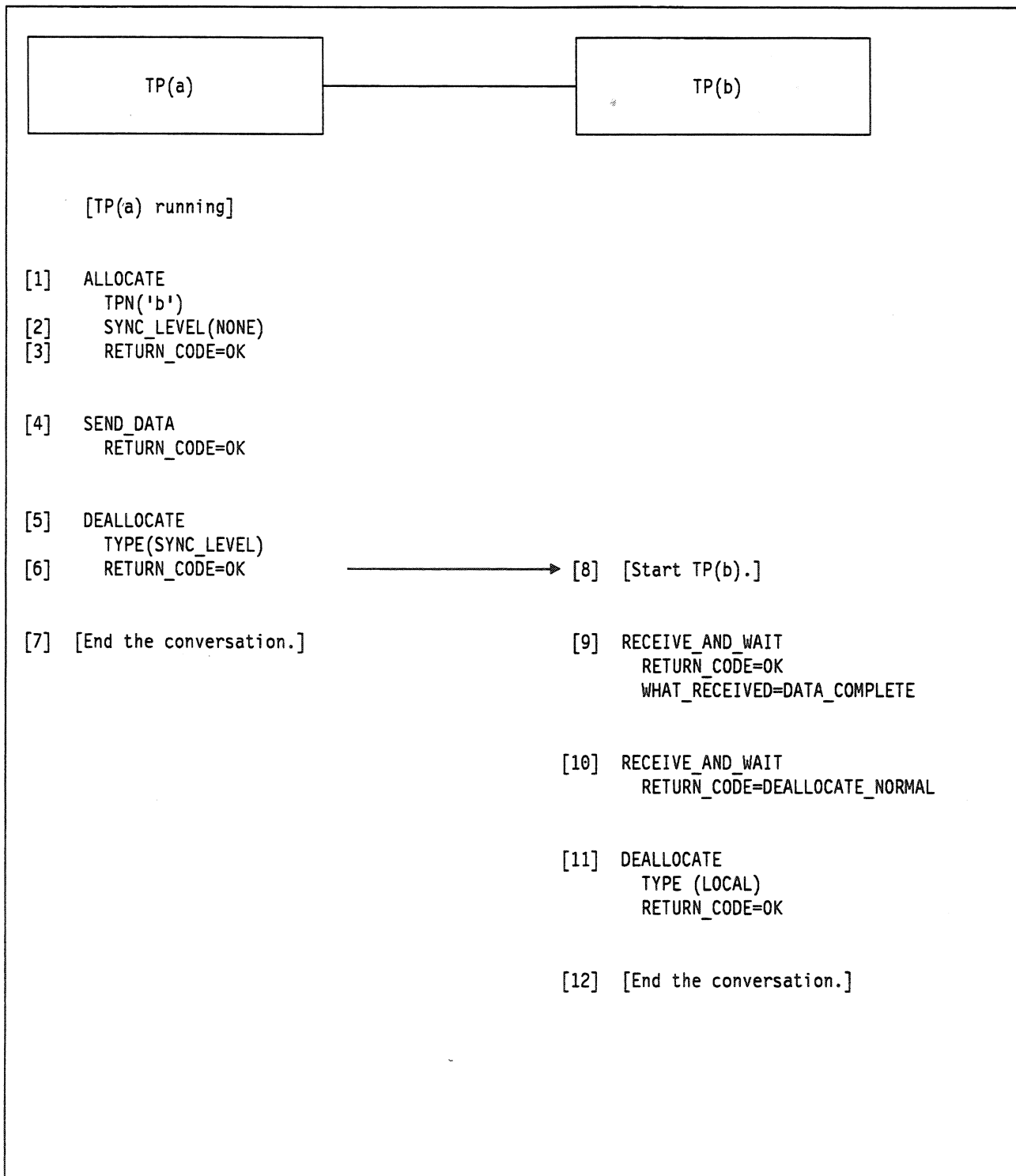


Figure B-1. ALLOCATE, SEND_DATA, DEALLOCATE—SYNC_LEVEL(NONE)

Notes for Figure B-1:

1. TP(a) issues `ALLOCATE` to request a conversation with partner program 'b', designated by the `TPN` parameter.
2. The `SYNC_LEVEL` parameter specifies a synchronization level of `NONE`, which means no confirmation or sync point processing.
3. The LU places the allocation request in its send buffer and returns control to TP(a) with the conversation in send state. Nothing is sent.
4. TP(a) issues `SEND_DATA`, causing the LU to place the data (a logical record) in its buffer behind the allocation request. Still, nothing is sent, because the LU does not yet have a sufficient amount of information for transmission (in this example).
5. TP(a) issues `DEALLOCATE` with `TYPE(SYNC_LEVEL)`, which implies no confirmation or sync point processing and therefore causes the LU to send the contents of its buffer together with a `DEALLOCATE_NORMAL` indication.
6. Because the synchronization level is `NONE`, `DEALLOCATE` with `TYPE(SYNC_LEVEL)` completes immediately and successfully. Contrast this with Figure B-2 on page B-6.
7. The conversation is deallocated from the session at the completion of `DEALLOCATE`.
8. TP(b) is started, with the conversation in receive state, when its LU receives the allocation request.
9. TP(b) issues `RECEIVE_AND_WAIT` and receives the complete logical record.
10. TP(b) issues another `RECEIVE_AND_WAIT` and receives the `DEALLOCATE_NORMAL` indication.
11. TP(b) issues `DEALLOCATE` with `TYPE(LOCAL)`, causing the LU to discard its control information for the conversation.
12. The conversation ends for TP(b).

Examples Using Basic Conversation Verbs

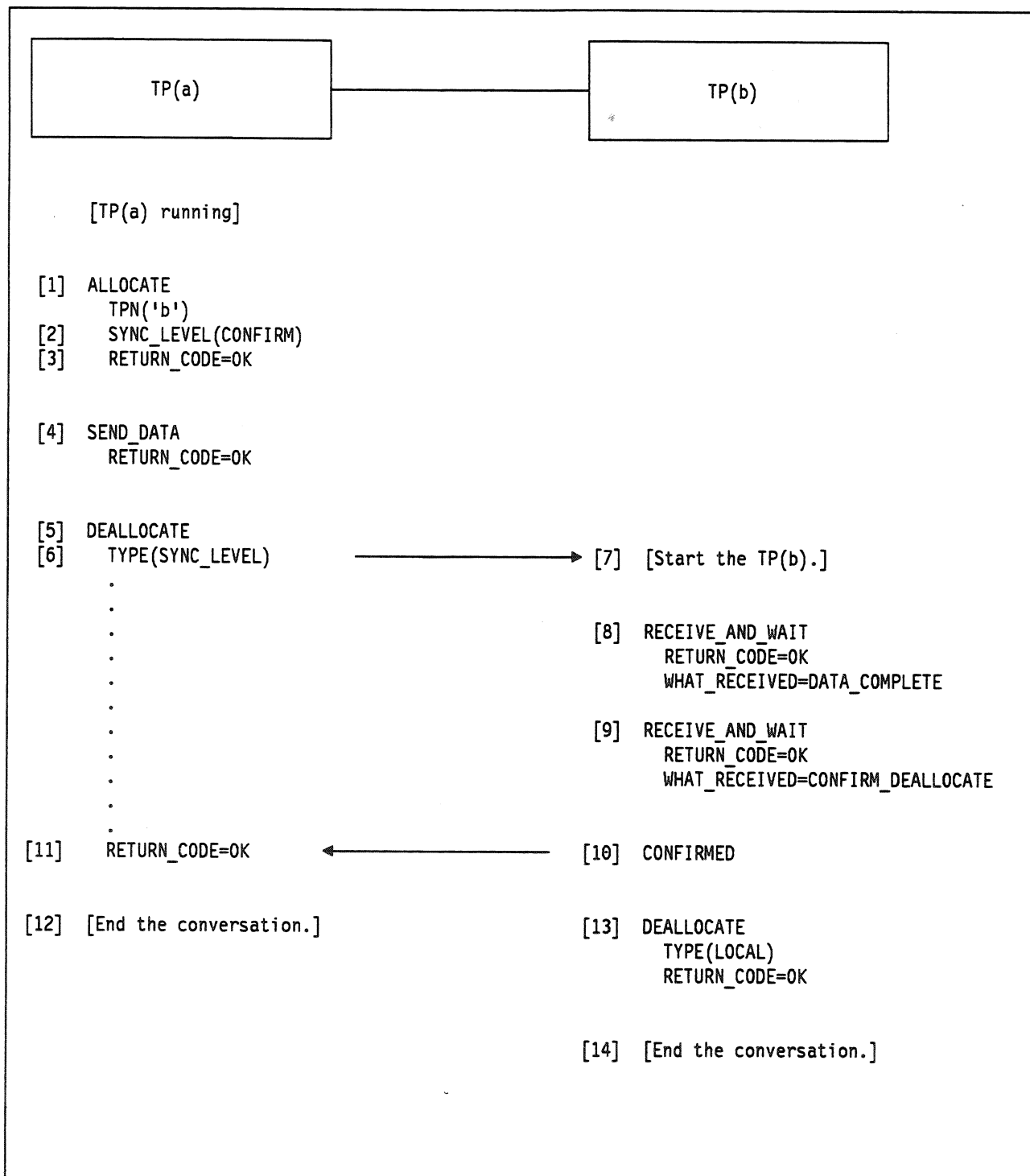


Figure B-2. ALLOCATE, SEND_DATA, DEALLOCATE—SYNC_LEVEL(CONFIRM)

Notes for Figure B-2:

1. TP(a) issues **ALLOCATE** to request a conversation with partner program 'b'.
2. The **SYNC_LEVEL** parameter specifies a synchronization level of **CONFIRM**, which means confirmation processing is permitted.
3. The LU places the allocation request in its send buffer and returns control to TP(a) with the conversation in send state. Nothing is sent.
4. TP(a) issues **SEND_DATA**, causing the LU to place the data (a logical record) in its buffer behind the allocation request. Still, nothing is sent.
5. TP(a) issues **DEALLOCATE** with **TYPE(SYNC_LEVEL)**, which implies confirmation processing and therefore causes the LU to send the contents of its buffer together with a **CONFIRM_DEALLOCATE** request.
6. Because the synchronization level is **CONFIRM**, **DEALLOCATE** with **TYPE(SYNC_LEVEL)** causes the LU to suspend TP(a)'s processing until it receives a response, affirmative or negative.
7. TP(b) is started, with the conversation in receive state, when its LU receives the allocation request.
8. TP(b) issues **RECEIVE_AND_WAIT** and receives the complete logical record.
9. TP(b) issues another **RECEIVE_AND_WAIT** and receives the **CONFIRM_DEALLOCATE** request.
10. TP(b) responds affirmatively by issuing **CONFIRMED**, causing its LU to send a positive response. If TP(b) had responded negatively by issuing **SEND_ERROR** (not shown), the conversation would remain allocated.
11. The LU returns control to TP(a), indicating an affirmative response and successful completion of the **DEALLOCATE**.
12. The conversation is deallocated from the session at the completion of **DEALLOCATE**.
13. TP(b) issues **DEALLOCATE** with **TYPE(LOCAL)**, causing the LU to discard its control information for the conversation.
14. The conversation ends for TP(b).

Examples Using Basic Conversation Verbs

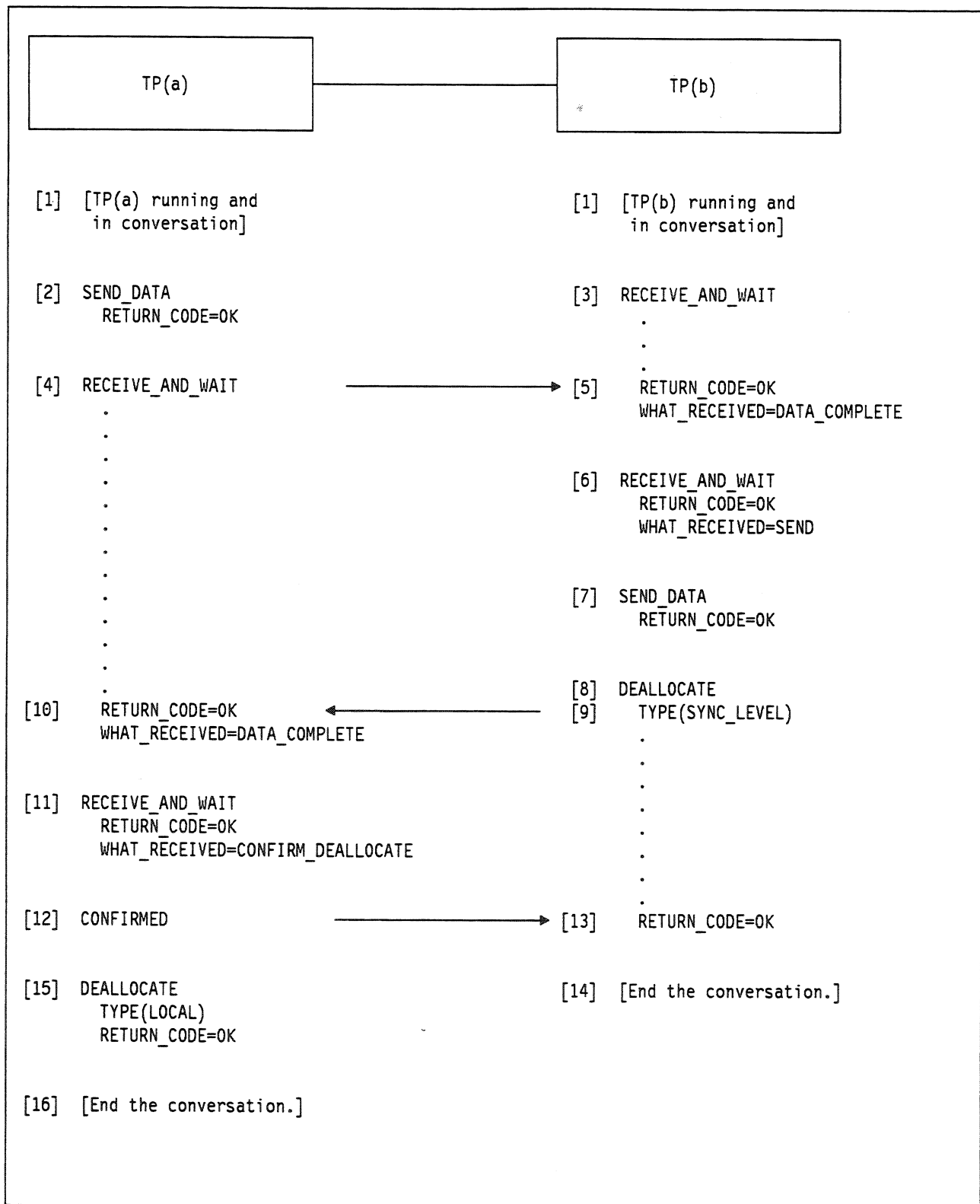


Figure B-3. RECEIVE_AND_WAIT, DEALLOCATE-SYNC_LEVEL(CONFIRM)

Notes for Figure B-3:

1. Assume a conversation with SYNC_LEVEL(CONFIRM) already exists between TP(a) and TP(b), and the conversation is now in send state for TP(a) and receive state for TP(b).
2. TP(a) issues SEND_DATA, causing the LU to place the data (a logical record) in its buffer. Nothing is sent.
3. TP(b) issues RECEIVE_AND_WAIT, causing the LU to suspend TP(b)'s processing until it receives information.
4. TP(a) issues RECEIVE_AND_WAIT, causing the LU to send the contents of its buffer together with the SEND indication. The LU suspends TP(a)'s processing until it receives information.
5. The LU returns control to TP(b), indicating that the program has received the complete logical record.
6. TP(b) issues another RECEIVE_AND_WAIT and receives the SEND indication.
7. TP(b) issues SEND_DATA, causing the LU to place the data (a logical record) in its buffer. Nothing is sent.
8. TP(b) issues DEALLOCATE with TYPE(SYNC_LEVEL), which causes the LU

to send the contents of its buffer together with a CONFIRM_DEALLOCATE request.

9. Because the synchronization level is CONFIRM, DEALLOCATE with TYPE(SYNC_LEVEL) causes the LU to suspend TP(b)'s processing until it receives a response, affirmative or negative.
10. The LU returns control to TP(a), indicating that the program has received the complete logical record.
11. TP(a) issues another RECEIVE_AND_WAIT and receives the CONFIRM_DEALLOCATE request.
12. TP(a) responds affirmatively by issuing CONFIRMED, causing its LU to send a positive response.
13. The LU returns control to TP(b), indicating an affirmative response and successful completion of the DEALLOCATE.
14. The conversation is deallocated from the session at the completion of DEALLOCATE.
15. TP(a) issues DEALLOCATE with TYPE(LOCAL), causing the LU to discard its control information for the conversation.
16. The conversation ends for TP(a).

Examples Using Basic Conversation Verbs

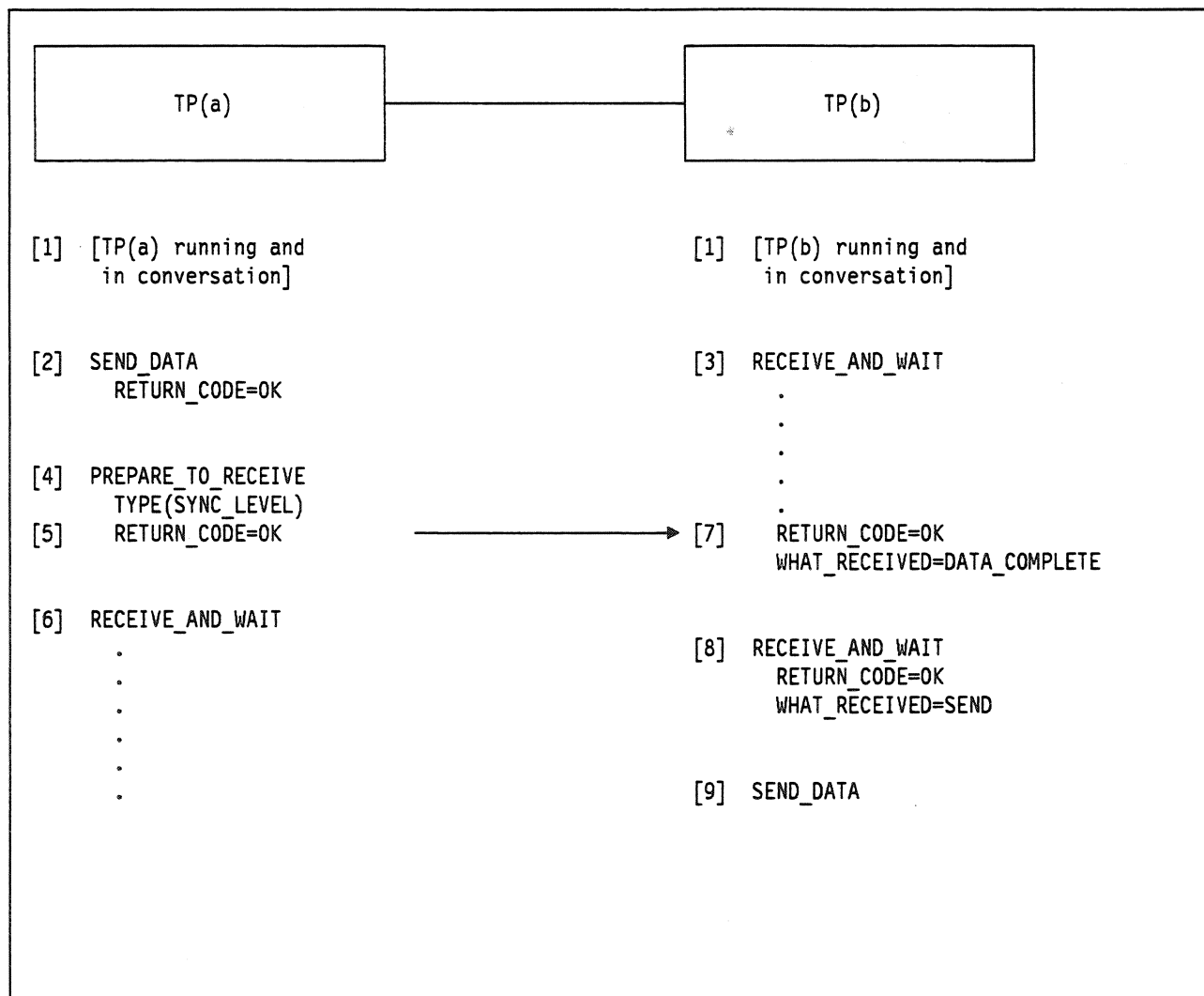


Figure B-4. `PREPARE_TO_RECEIVE-SYNC_LEVEL(NONE)`

Notes for Figure B-4:

1. Assume a conversation with `SYNC_LEVEL(NONE)` already exists between TP(a) and TP(b), and the conversation is now in send state for TP(a) and receive state for TP(b).
2. TP(a) issues `SEND_DATA`, causing the LU to place the data (a logical record) in its buffer. Nothing is sent.
3. TP(b) issues `RECEIVE_AND_WAIT`, causing the LU to suspend TP(b)'s processing until it receives information.
4. TP(a) issues `PREPARE_TO_RECEIVE` with `TYPE(SYNC_LEVEL)`, which causes the LU to send the contents of its buffer together with the `SEND` indication.
5. Because the synchronization level is `NONE`, `PREPARE_TO_RECEIVE` with `TYPE(SYNC_LEVEL)` completes immediately and successfully. Contrast this with Figure B-5 on page B-12.
6. The conversation for TP(a) is now in receive state, so TP(a) issues a `RECEIVE_AND_WAIT`.
7. The LU returns control to TP(b), indicating that the program has received the complete logical record.
8. TP(b) issues another `RECEIVE_AND_WAIT` and receives the `SEND` indication.
9. The conversation for TP(b) is now in send state, so TP(b) issues a `SEND_DATA`.

Examples Using Basic Conversation Verbs

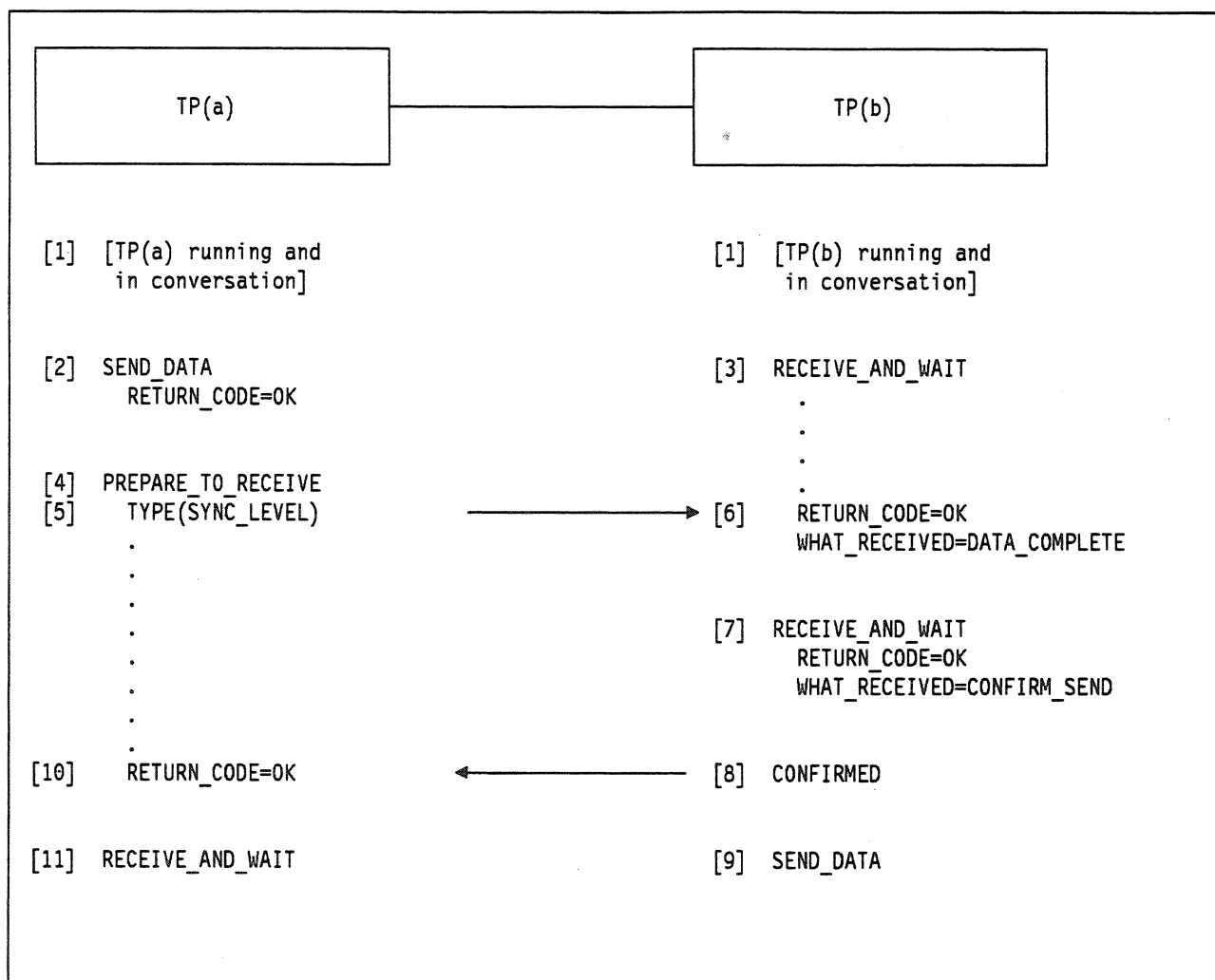


Figure B-5. PREPARE_TO_RECEIVE-SYNC_LEVEL(CONFIRM)

Notes for Figure B-5:

1. Assume a conversation with SYNC_LEVEL(CONFIRM) already exists between TP(a) and TP(b), and the conversation is now in send state for TP(a) and receive state for TP(b).
2. TP(a) issues SEND_DATA, causing the LU to place the data (a logical record) in its buffer. Nothing is sent.
3. TP(b) issues RECEIVE_AND_WAIT, causing the LU to suspend TP(b)'s processing until it receives information.
4. TP(a) issues PREPARE_TO_RECEIVE with TYPE(SYNC_LEVEL), which causes the LU to send the contents of its buffer together with the CONFIRM_SEND request.
5. Because the synchronization level is CONFIRM, PREPARE_TO_RECEIVE with TYPE(SYNC_LEVEL) causes the LU to suspend TP(a)'s processing until it receives a response, affirmative or negative.
6. The LU returns control to TP(b), indicating that the program has received the complete logical record.
7. TP(b) issues another RECEIVE_AND_WAIT and receives the CONFIRM_SEND request.
8. TP(b) responds affirmatively by issuing CONFIRMED, causing its LU to send a positive response.
9. The conversation for TP(b) is now in send state, so TP(b) issues a SEND_DATA.
10. The LU returns control to TP(a), indicating an affirmative response and successful completion of the PREPARE_TO_RECEIVE.
11. The conversation for TP(a) is now in receive state, so TP(a) issues a RECEIVE_AND_WAIT.

Examples Using Basic Conversation Verbs

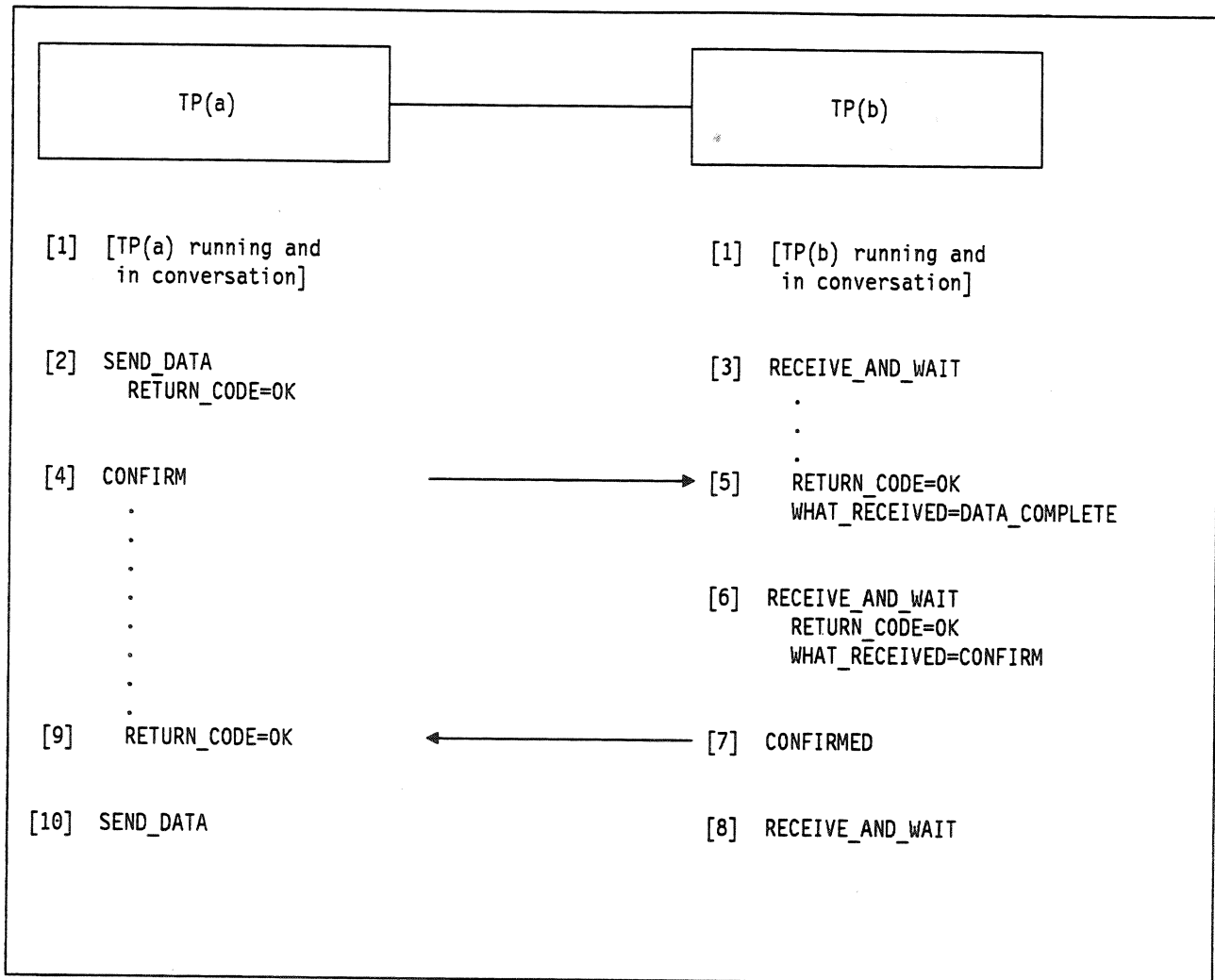


Figure B-6. CONFIRM

Notes for Figure B-6:

1. Assume a conversation with SYNC_LEVEL(CONFIRM) already exists between TP(a) and TP(b), and the conversation is now in send state for TP(a) and receive state for TP(b).
2. TP(a) issues SEND_DATA, causing the LU to place the data (a logical record) in its buffer. Nothing is sent.
3. TP(b) issues RECEIVE_AND_WAIT, causing the LU to suspend TP(b)'s processing until it receives information.
4. TP(a) issues CONFIRM in order to synchronize the processing of the two programs. The CONFIRM verb causes the LU to send the contents of its buffer together with the CONFIRM request. The LU suspends TP(a)'s processing until it receives a response, affirmative or negative.
5. The LU returns control to TP(b), indicating that the program has received the complete logical record.
6. TP(b) issues another RECEIVE_AND_WAIT and receives the CONFIRM request.
7. TP(b) responds affirmatively by issuing CONFIRMED, causing its LU to send a positive response.
8. The conversation for TP(b) is still in receive state, so TP(b) issues another RECEIVE_AND_WAIT.
9. The LU returns control to TP(a), indicating an affirmative response and successful completion of the CONFIRM.
10. The conversation for TP(a) is still in send state, so TP(a) issues another SEND_DATA.

Examples Using Basic Conversation Verbs

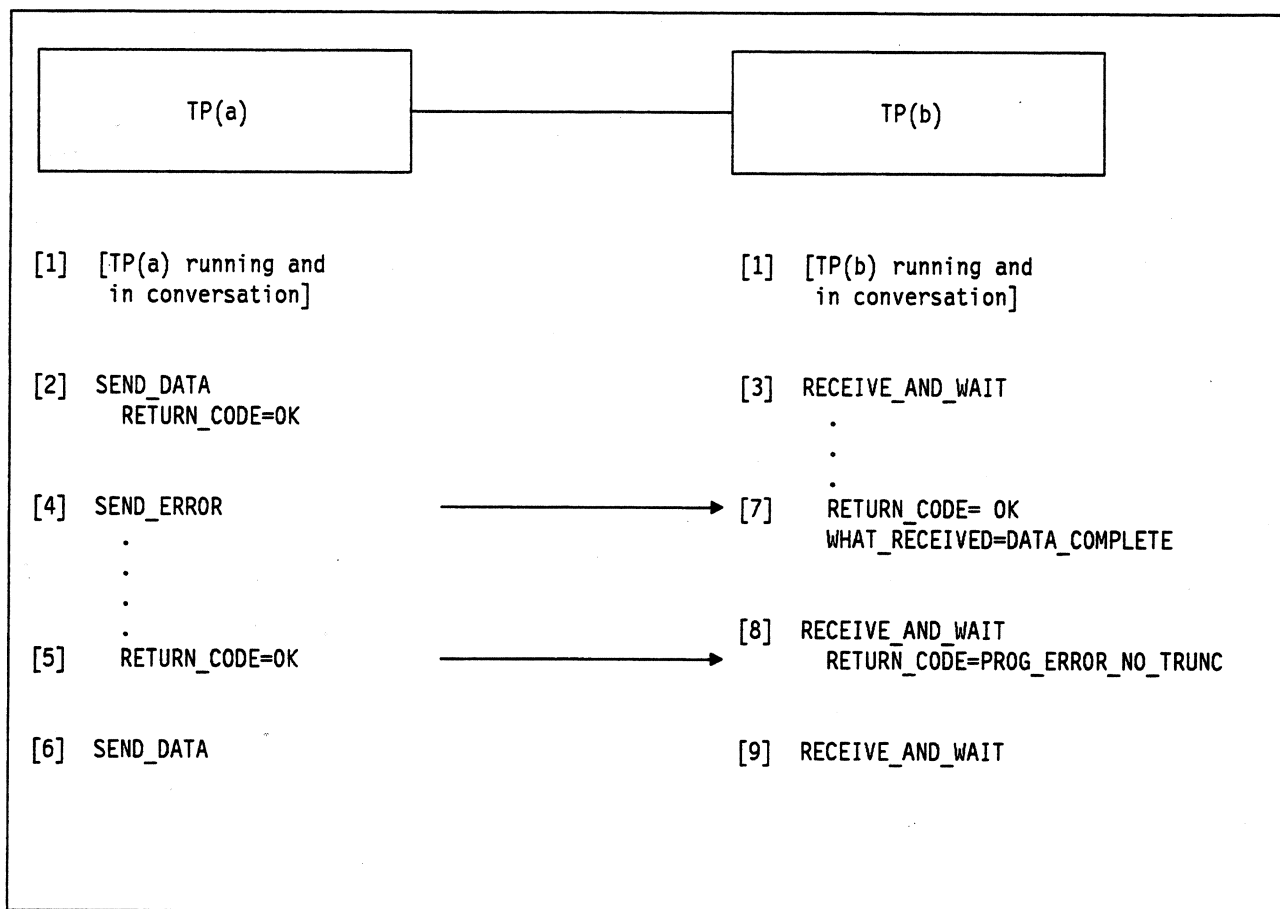


Figure B-7. SEND_ERROR in Send State

Examples Using Basic Conversation Verbs

Notes for Figure B-7:

1. Assume a conversation already exists between TP(a) and TP(b), and the conversation is now in send state for TP(a) and receive state for TP(b).
2. TP(a) issues SEND_DATA, causing the LU to place the data (a logical record) in its buffer. Nothing is sent.
3. TP(b) issues RECEIVE_AND_WAIT, causing the LU to suspend TP(b)'s processing until it receives information.
4. TP(a) issues SEND_ERROR in order to notify the partner program of an error. The SEND_ERROR verb causes the LU to send the contents of its buffer.
5. After sending the contents of its buffer, the LU sends the error notification and returns control to the program.
6. The conversation for TP(a) is still in send state, so TP(a) issues another SEND_DATA, possibly containing additional error-recovery information.
7. The LU returns control to TP(b), indicating that the program has received the complete logical record.
8. TP(b) issues another RECEIVE_AND_WAIT and receives the error notification, PROG_ERROR_NO_TRUNC, meaning TP(a) issued a SEND_ERROR that did not truncate the logical record it sent.
9. The conversation for TP(b) is still in receive state, so TP(b) issues another RECEIVE_AND_WAIT.

Examples Using Basic Conversation Verbs

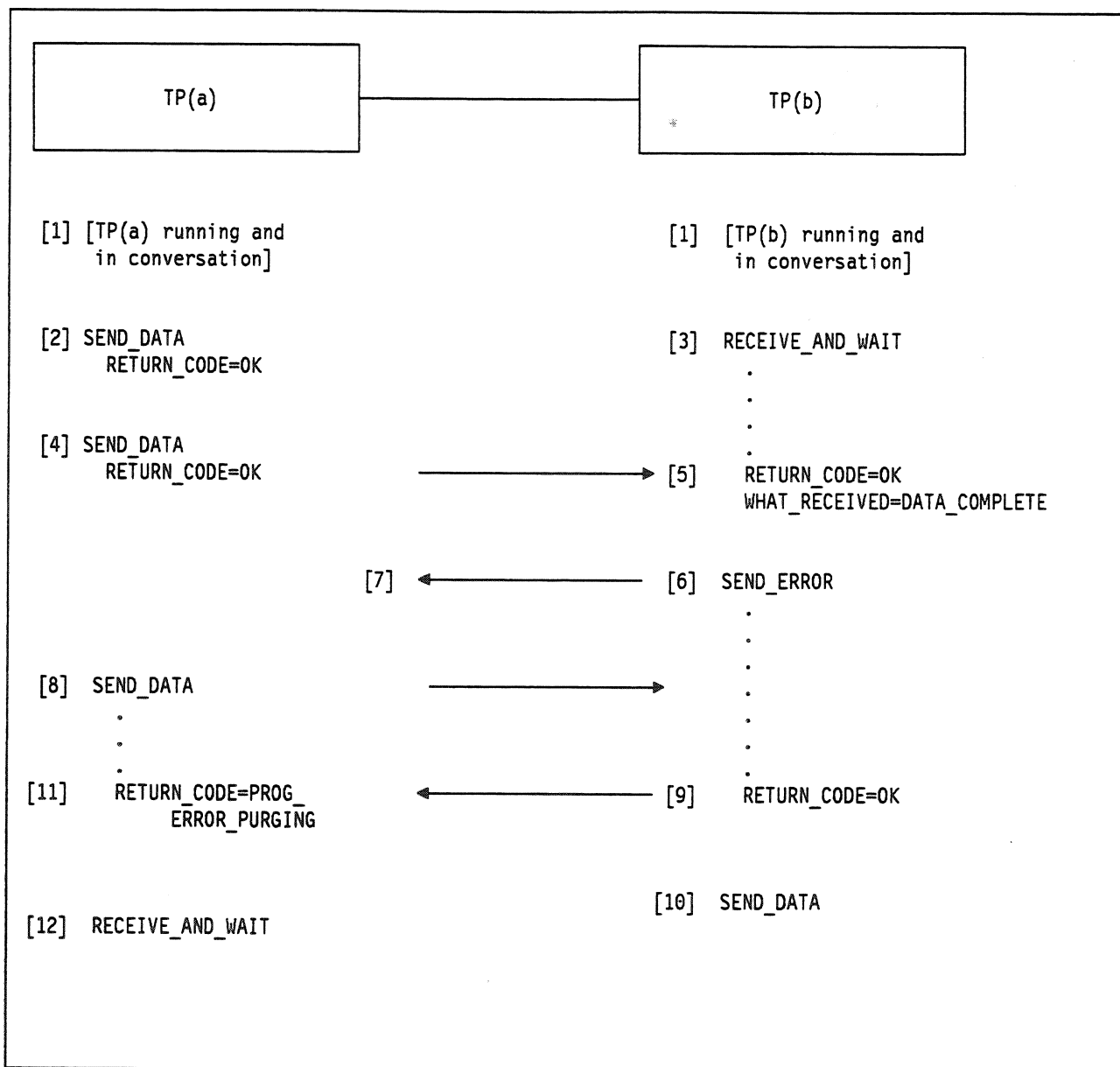


Figure B-8. `SEND_ERROR` in Receive State

Notes for Figure B-8:

1. Assume a conversation already exists between TP(a) and TP(b), and the conversation is now in send state for TP(a) and receive state for TP(b).
2. TP(a) issues SEND_DATA, causing the LU to place the data (a logical record) in its buffer. Nothing is sent.
3. TP(b) issues RECEIVE_AND_WAIT, causing the LU to suspend TP(b)'s processing until it receives information.
4. TP(a) issues another SEND_DATA, causing the LU to place the data (another logical record) in its buffer. The LU now has more than enough data for transmission, so it sends some of the contents of its buffer, and retains the remainder for later transmission.
5. The LU returns control to TP(b), indicating that the program has received a complete logical record.
6. TP(b) issues SEND_ERROR in order to notify the partner program of an error. The SEND_ERROR verb causes the LU to purge information it has received and not yet passed to TP(b), and to send a negative response. The LU then suspends TP(b)'s processing awaiting the receipt of SEND control, and continues purging information until SEND control arrives.
7. The LU for TP(a) receives the negative response, causing it to purge the remaining contents of its buffer.
8. TP(a) issues SEND_DATA. The SEND_DATA is unsuccessful—the LU does not place the data in its buffer. The SEND_DATA causes the LU to send the SEND control without data. The LU then suspends TP(a)'s processing awaiting the receipt of the error notification.
9. The LU for TP(b) receives the SEND control, sends the error notification, and returns control to TP(b).
10. The conversation for TP(b) is now in send state, so TP(b) issues a SEND_DATA, possibly containing additional error-recovery information.
11. The LU returns control to TP(a), indicating that it has received the error notification, PROG_ERROR_PURGING.
12. The conversation for TP(a) is now in receive state, so TP(a) issues a RECEIVE_AND_WAIT.

Examples Using Basic Conversation Verbs

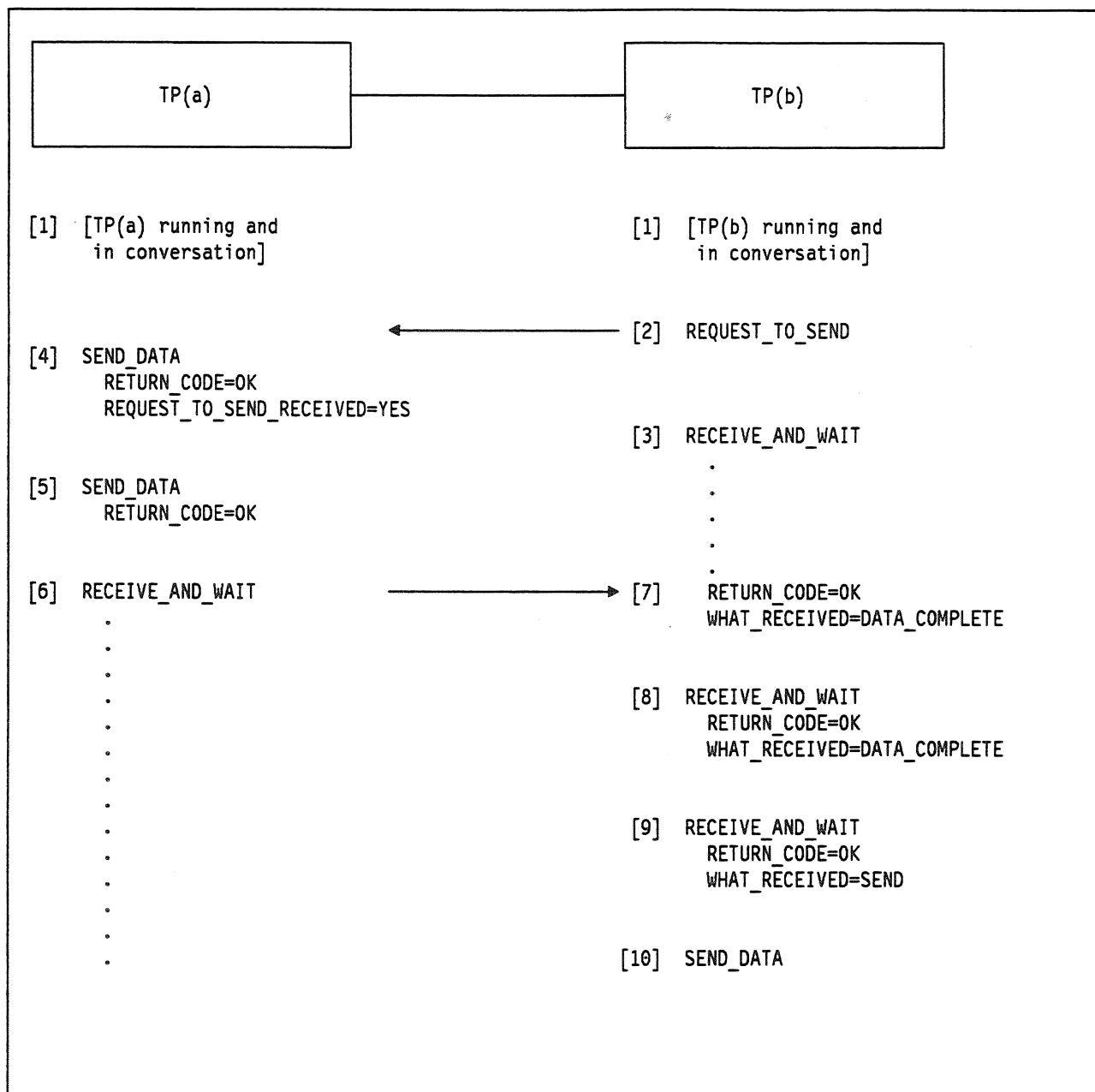


Figure B-9. REQUEST_TO_SEND

Notes for Figure B-9:

1. Assume a conversation already exists between TP(a) and TP(b), and the conversation is now in send state for TP(a) and receive state for TP(b).
2. TP(b) issues REQUEST_TO_SEND and the LU sends the REQUEST_TO_SEND indication. The conversation for TP(b) remains in receive state because REQUEST_TO_SEND does not force TP(a) to relinquish SEND control. Contrast this with SEND_ERROR in Figure B-8 on page B-18.
3. TP(b) issues RECEIVE_AND_WAIT, causing the LU to suspend TP(b)'s processing until it receives information.
4. TP(a) issues SEND_DATA, causing the LU to place the data (a logical record) in its buffer and to indicate to TP(a) that it has received a REQUEST_TO_SEND indication. Nothing is sent.
5. TP(a) issues another SEND_DATA, causing the LU to place the data (another logical record) in its buffer. The LU still does not have enough data for transmission, so nothing is sent.
6. TP(a) issues RECEIVE_AND_WAIT, causing the LU to send the contents of its buffer together with the SEND indication. The LU suspends TP(a)'s processing until it receives information.
7. The LU returns control to TP(b), indicating that the program has received a complete logical record.
8. TP(b) issues another RECEIVE_AND_WAIT and receives another complete logical record.
9. TP(b) issues another RECEIVE_AND_WAIT and receives the SEND indication.
10. The conversation for TP(b) is now in send state, so TP(b) issues a SEND_DATA.

Examples Using Basic Conversation Verbs

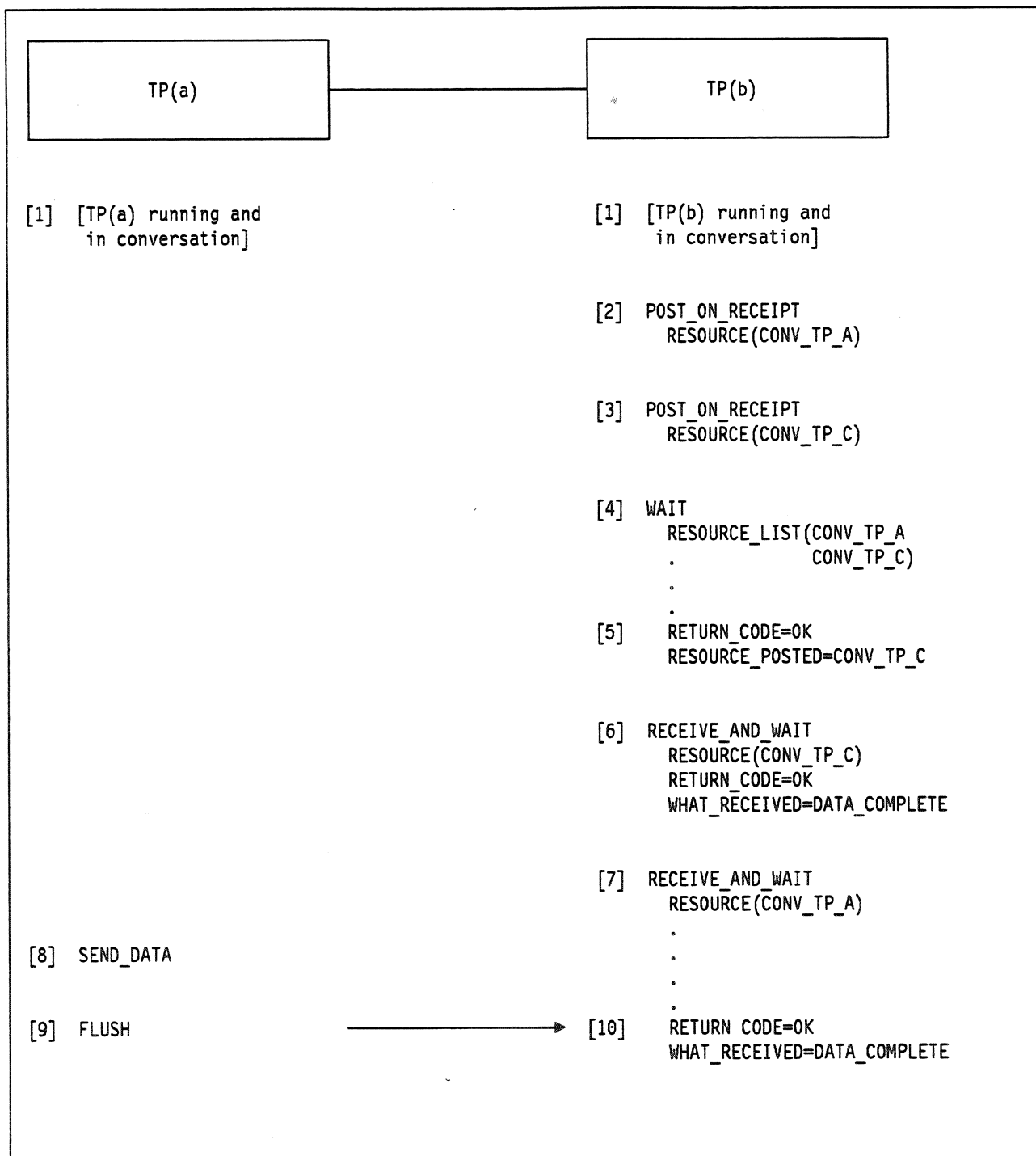


Figure B-10. POST_ON_RECEIPT, WAIT

Notes for Figure B-10:

1. Assume a conversation already exists between TP(a) and TP(b), and the conversation is now in send state for TP(a) and receive state for TP(b). Also assume a conversation already exists between TP(b) and TP(c), not shown, and this conversation is in receive state for TP(b).
2. TP(b) issues POST_ON_RECEIPT for the conversation with TP(a).
3. TP(b) issues another POST_ON_RECEIPT for the conversation with TP(c).
4. TP(b) then issues WAIT with a resource list specifying both conversations. In this way, the program can receive the information on the conversation on which the information arrives first. The WAIT causes the LU to suspend TP(b)'s processing until it receives information on any of the specified conversations.
5. Information arrives on the conversation with TP(c). The LU resets the posting on that conversation and returns control to TP(b), indicating to TP(b) which conversation was posted.
6. TP(b) issues RECEIVE_AND_WAIT for the conversation with TP(c) and receives a complete logical record.
7. TP(b) then issues RECEIVE_AND_WAIT for the conversation with TP(a), causing the LU to suspend TP(b)'s processing until it receives information on that conversation.
8. TP(a) issues SEND_DATA, causing the LU to place the data (a logical record) in its buffer. Nothing is sent.
9. TP(a) issues FLUSH, causing the LU to send the contents of its buffer.
10. The LU returns control to TP(b), indicating that the program has received a complete logical record.

Examples Using Basic Conversation Verbs

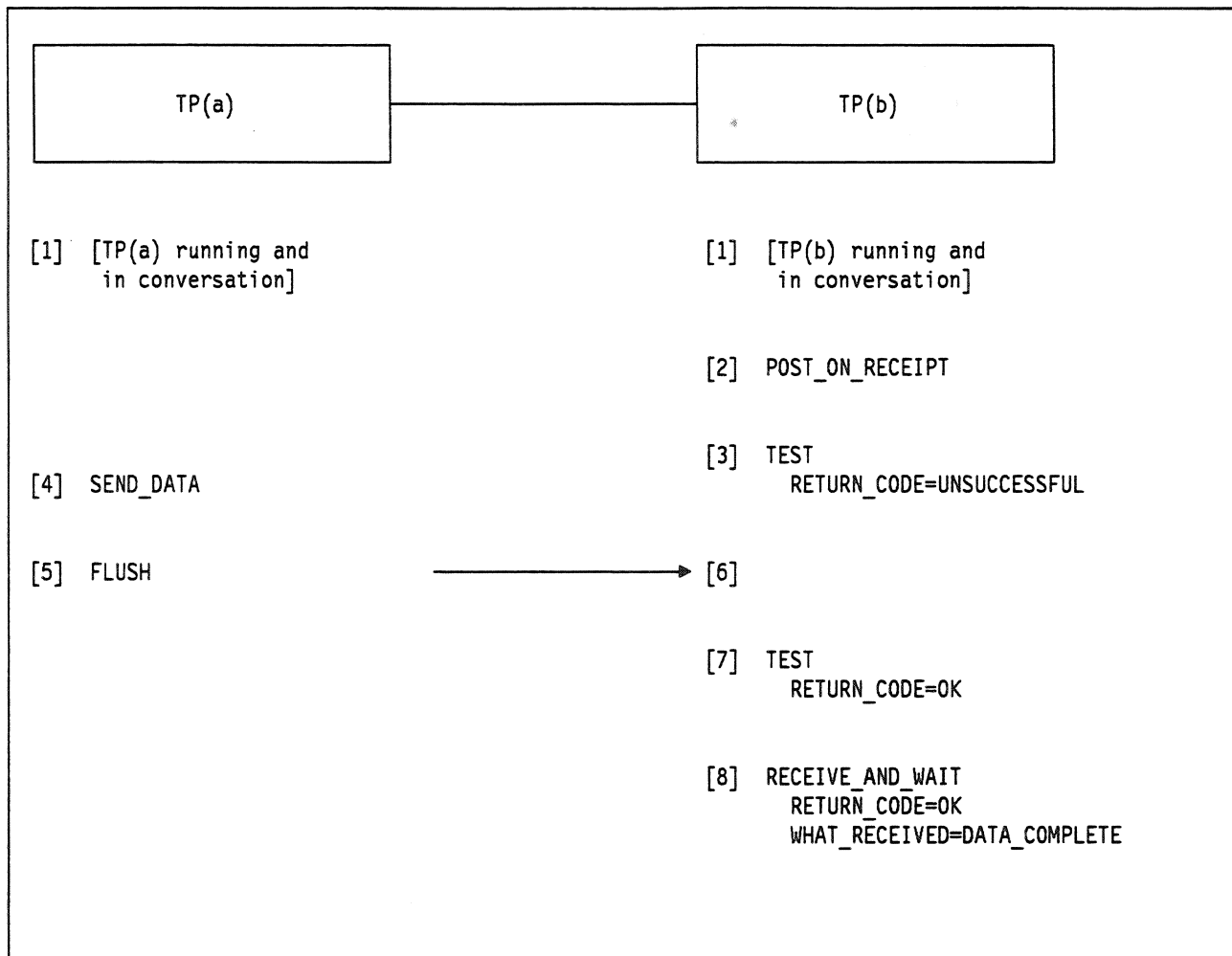


Figure B-11. POST_ON_RECEIPT, TEST

Examples Using Basic Conversation Verbs

Notes for Figure B-11:

1. Assume a conversation already exists between TP(a) and TP(b), and the conversation is now in send state for TP(a) and receive state for TP(b).
2. TP(b) issues POST_ON_RECEIPT for the conversation with TP(a).
3. TP(b) then issues TEST, which returns with RETURN_CODE = UNSUCCESSFUL indicating information is not available. Posting remains active and TP(b) continues processing.
4. TP(a) issues SEND_DATA, causing the LU to place the data (a logical record) in its buffer. Nothing is sent.
5. TP(a) issues FLUSH, causing the LU to send the contents of its buffer.
6. Data arrives on the conversation, causing the LU to post the event.
7. Some time later, TP(b) issues TEST again. This time it returns with RETURN_CODE = OK, and posting is reset.
8. TP(b) issues RECEIVE_AND_WAIT for the conversation and receives a complete logical record.

Examples Using Basic Conversation Verbs

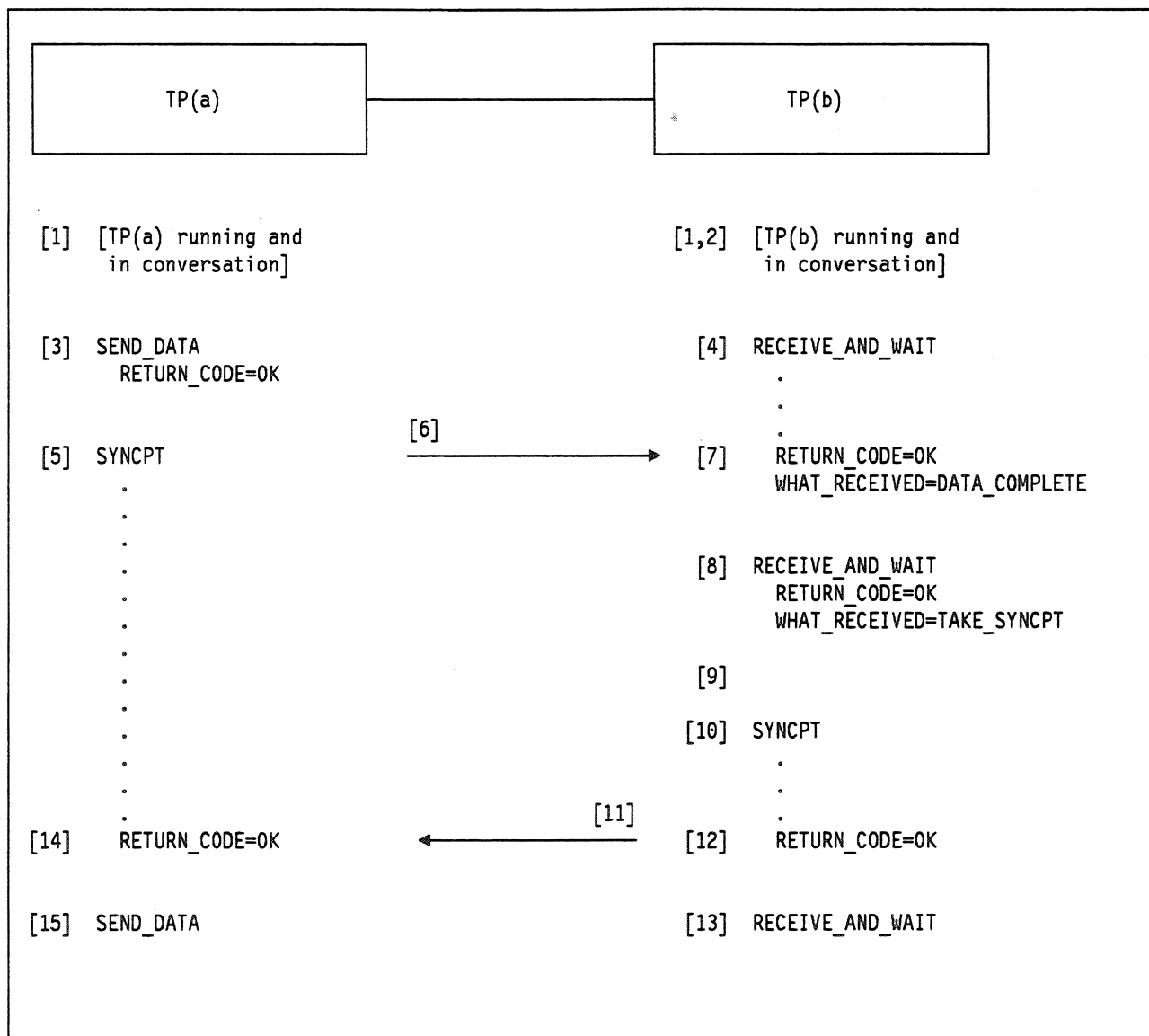


Figure B-12. SYNCPT

Notes for Figure B-12:

1. Assume a conversation with SYNC_LEVEL(SYNCPT) already exists between TP(a) and TP(b), and the conversation is now in send state for TP(a) and receive state for TP(b).
2. Assume one or more conversations with SYNC_LEVEL(SYNCPT) already exist between TP(b) and other programs.
3. TP(a) issues SEND_DATA, causing the LU to place the data (a logical record) in its buffer. Nothing is sent.
4. TP(b) issues RECEIVE_AND_WAIT, causing the LU to suspend TP(b)'s processing until it receives information.
5. TP(a) issues SYNCPT in order to advance all protected resources throughout the distributed logical unit of work to the next synchronization point. The LU suspends TP(a)'s processing until the sync point processing is complete. As part of the sync point processing, the LUs send and receive commands on the conversations; the commands are referred to in this example as a sync point request and reply. These commands are not apparent to the programs.
6. The SYNCPT verb causes the LU to send the contents of its buffer together with the initial sync point request.
7. The LU for TP(b) receives the data and sync point request. The LU returns control to TP(b), indicating that the program has received a complete logical record.
8. TP(b) issues another RECEIVE_AND_WAIT and receives the TAKE_SYNCPT request, which is what the LU indicates to the program as a result of receiving the sync point request.
9. TP(b) finishes processing of protected local resources, if necessary, and ensures all other protected conversations are in send state.
10. TP(b) issues SYNCPT, causing the LU to send the contents of its buffers (one for each conversation) together with a sync point

request on all other protected conversations. The LU suspends TP(b)'s processing until a sync point reply is received on all these conversations.

11. After receiving sync point replies on all of the other protected conversations, the LU for TP(b) sends a sync point reply on the conversation on which it received the initial sync point request.
12. The LU returns control to TP(b) indicating successful completion of the SYNCPT for all protected resources allocated to TP(b) and all "down stream" TPs, that is, to all TPs other than TP(a).
13. The conversation for TP(b) is still in receive state, so TP(b) issues another RECEIVE_AND_WAIT.

Note that the state of the conversation at TP(b) following the successful sync point operation depends on the value of the WHAT_RECEIVED indicator in step 8. If the TP received TAKE_SYNCPT_SEND, it would now be in send state; if it received TAKE_SYNCPT_DEALLOCATE, it would now be in deallocate state.

14. The LU for TP(a) receives the final sync point reply and returns control to TP(a) indicating successful completion of the SYNCPT for all protected resources throughout the distributed logical unit of work.
15. The conversation for TP(a) is still in send state, so TP(a) issues another SEND_DATA.

Note that the state of the conversation at TP(a) following the successful sync point operation depends on the verb preceding the SYNCPT verb. If the verb were PREPARE_TO_RECEIVE TYPE(SYNC_LEVEL), the state would now be receive. If the verb were DEALLOCATE TYPE(SYNC_LEVEL), the state would now be reset.

Note: More sync point commands may actually be exchanged between the participating LUs than the flows in this example indicate. See *SNA LU 6.2 Reference: Peer Protocols* for details.

Examples Using Basic Conversation Verbs

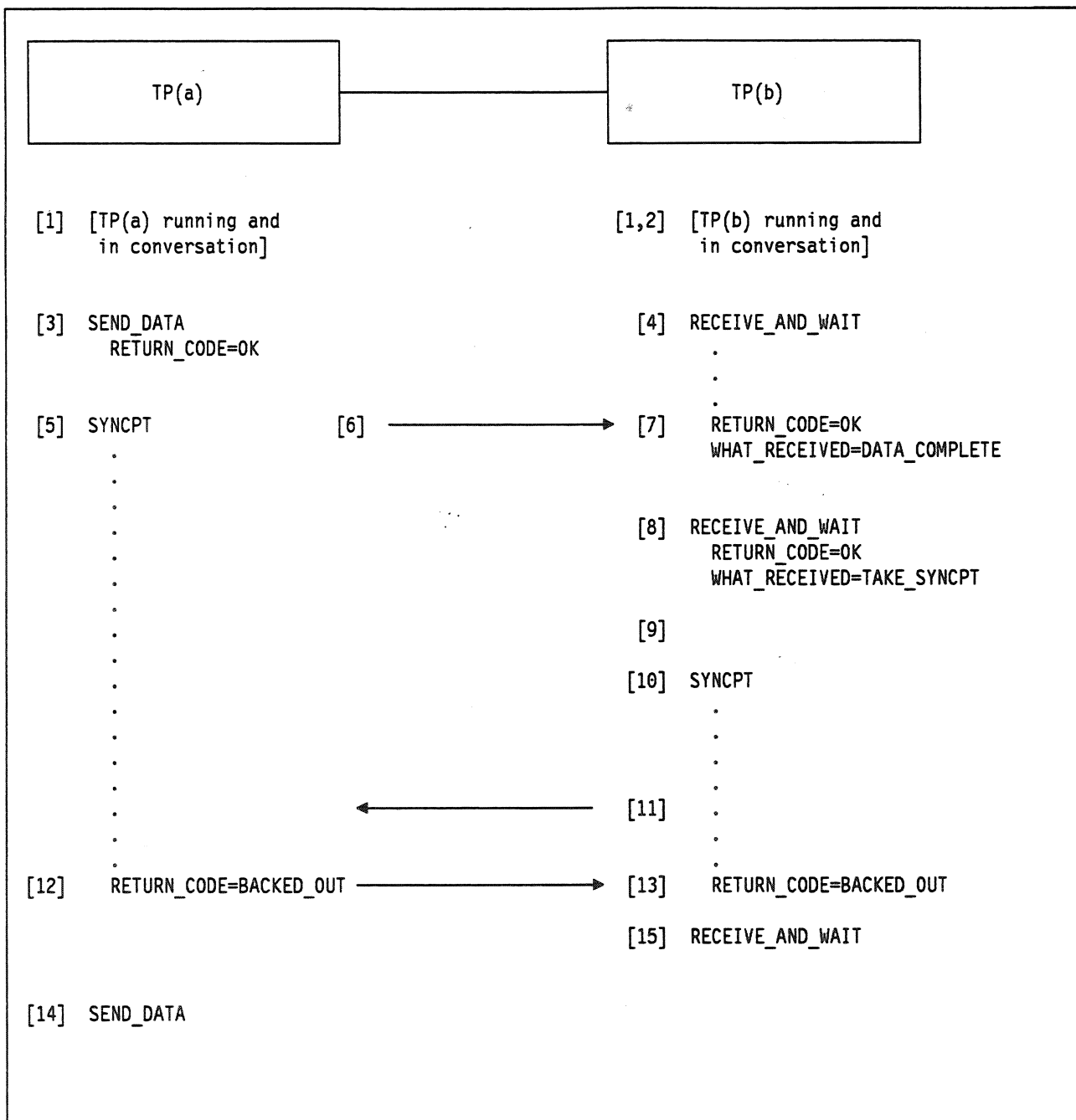


Figure B-13. SYNCPT, BACKOUT

Notes for Figure B-13:

1. Assume a conversation with SYNC_LEVEL(SYNCPT) already exists between TP(a) and TP(b), and the conversation is now in send state for TP(a) and receive state for TP(b).
2. Assume one or more conversations with SYNC_LEVEL(SYNCPT) already exist between TP(b) and other programs.
3. TP(a) issues SEND_DATA, causing the LU to place the data (a logical record) in its buffer. Nothing is sent.
4. TP(b) issues RECEIVE_AND_WAIT, causing the LU to suspend TP(b)'s processing until it receives information.
5. TP(a) issues SYNCPT in order to advance all protected resources throughout the distributed logical unit of work to the next synchronization point. The LU suspends TP(a)'s processing until the sync point processing is complete. As part of the sync point processing, the LUs send and receive commands on the conversations; the commands are referred to in this example as a sync point request and reply. These commands are not apparent to the programs.
6. The SYNCPT verb causes the LU to send the contents of its buffer together with the initial sync point request.
7. The LU for TP(b) receives the data and sync point request. The LU returns control to TP(b), indicating that the program has received a complete logical record.
8. TP(b) issues another RECEIVE_AND_WAIT and receives the TAKE_SYNCPT request, which is what the LU indicates to the program as a result of receiving the sync point request.
9. TP(b) finishes processing of protected local resources, if necessary, and ensures all subordinate conversations are in send state.
10. TP(b) issues SYNCPT, causing the LU to send the contents of its buffers (one for each conversation) together with a sync point request on all other protected conversations. The LU suspends TP(b)'s processing until the

sync point processing on all these conversations is complete.

11. Instead of receiving sync point replies on all of the other protected conversations, the LU for TP(b) receives at least one BACKED_OUT indication. The LU at TP(b) restores protected local resources to the last synchronization point, and sends a backout message on all protected conversations except the one(s) on which it received the preceding BACKED_OUT indication. BACKOUT is accomplished throughout the distributed logical unit of work in the same manner as for TP(b). That is, the LU for each program receives a BACKED_OUT indication, and restores protected local resources to the last synchronization point and sends backout messages on all remaining protected conversations, if any. When the LU sends a backout message to another LU, such as in this figure, the LU of TP(a), it suspends waiting for a response.
12. The LU for TP(a) receives the BACKED_OUT indication, backs out local resources, propagates the backout message to other protected conversations, sends back a positive response, and returns control to TP(a) with the BACKED_OUT indication. It is unnecessary for the TP to issue the BACKOUT verb, since the local resources have already been restored to the last synchronization point, and the backout message has already been propagated to all other protected conversations.
13. When the LU at TP(b) receives the response to the backout message, it returns control to TP(b) with the BACKED_OUT indication. It is unnecessary for the TP to issue the BACKOUT verb, since the local resources have already been restored to the last synchronization point, and the backout message has already been propagated to all other conversations allocated with SYNC_LEVEL(SYNCPT).
14. The conversation for TP(a) is restored to send state—the state at the completion of the last synchronization point—so TP(a) issues another SEND_DATA, possibly containing error-recovery information.

Examples Using Basic Conversation Verbs

15. The conversation for TP(b) is restored to receive state—the state at the completion of the last synchronization point—so TP(b) issues another RECEIVE_AND_WAIT.

Note: More sync point commands may actually be exchanged between the participating LUs than the flows in this example indicate. See *SNA LU 6.2 Reference: Peer Protocols* for details.

Nonblocking Examples

The nonblocking option brings additional functions and support to the programming interface. This section describes, from the application programmer's point of view, how the nonblocking interface can be applied. The examples show the interactions only between the TP and the LU in a local system, since nonblocking support is merely a local function.

Figure B-14 on page B-32 presents an example of the issuance and completion of multiple nonblocking verbs. Special attention is needed in reusing the parameters on a nonblocking verb in a subsequent verb invocation. Abnormal use of a returned parameter (e.g., the wait object or return code variable) can result in an unpredictable outcome, as illustrated in Figure B-15 on page B-34. This abnormality can vary from one conversation type to another, as shown in Figure B-16 on page B-36. A rule of thumb is to reuse a parameter only when it is read-only.

Race conditions can occur when the returned parameters on the nonblocking verb are being filled by the LU. Figure B-17 on page B-38 shows an example of such a race condition. In order to prevent the complication caused by such race conditions, a properly implemented LU fills in the return code after filling in all the other returned parameters on the verb and then posts the associated wait object. The transaction program can reuse a wait object only when it has been identified as being posted.

Finally, the usefulness of the NULL wait object is illustrated by an example of bulk data transfer in Figure B-18 on page B-40.

Examples Using Basic Conversation Verbs

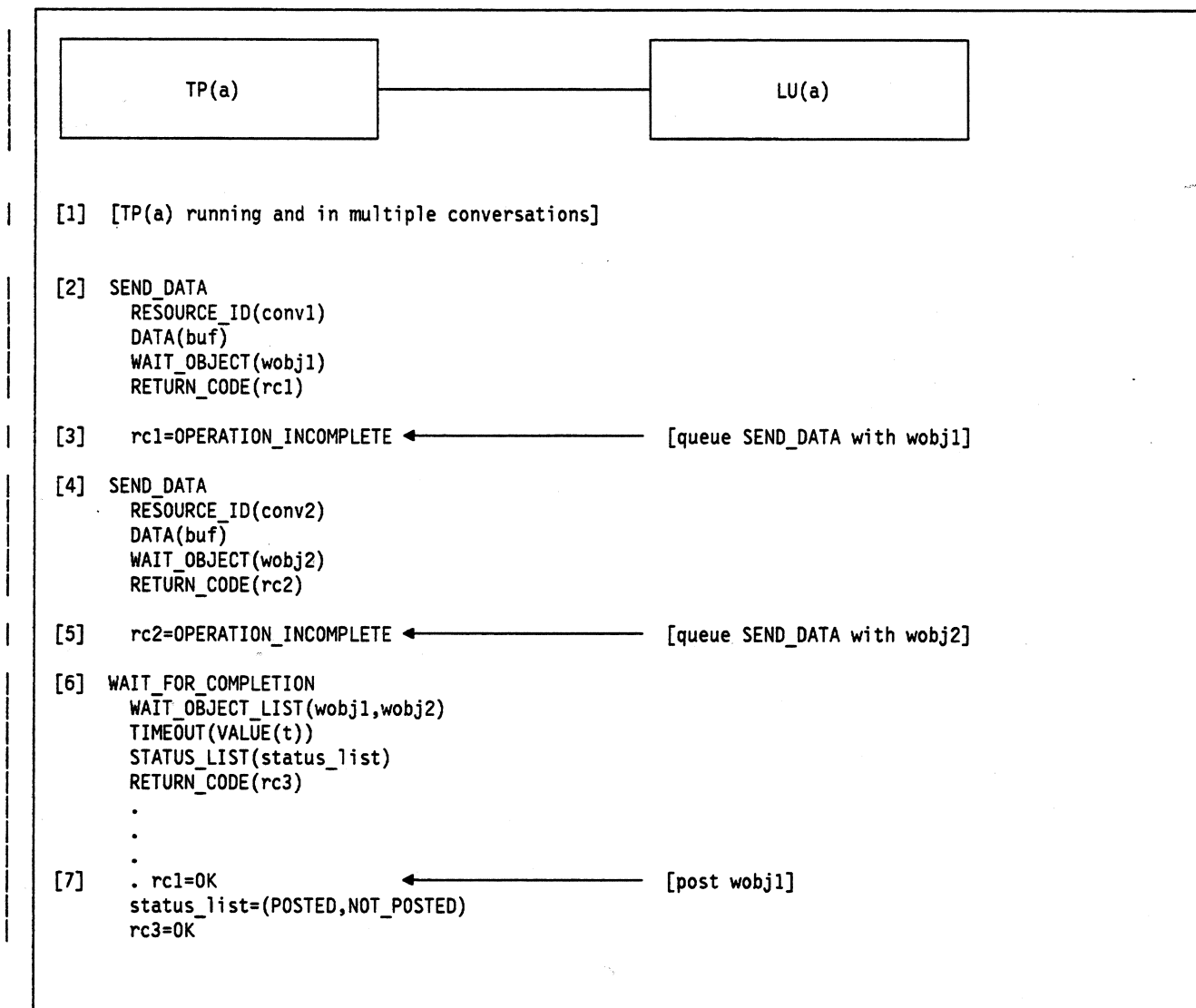


Figure B-14. Multiple Nonblocking Verb Operations

Examples Using Basic Conversation Verbs

Notes for Figure B-14:

1. Assume TP(a) is in conversations with two partner TPs, and is now in send state for both conversations.
2. TP(a) issues SEND_DATA to its first conversation partner in a nonblocking manner.
3. LU(a) suspends the verb operation because of some resource shortage, and returns control to TP(a) with an indication of OPERATION_INCOMPLETE. (The LU will later set the appropriate completion indication in the same return code variable.)
4. TP(a) subsequently issues another SEND_DATA to its second conversation partner in a nonblocking manner.
5. LU(a) suspends the verb operation again because of resource shortage, and returns

control to TP(a) with an indication of OPERATION_INCOMPLETE.

6. TP(a) monitors the status of the two outstanding operations by issuing WAIT_FOR_COMPLETION.
7. LU(a) completes the first request, sets the return code, rc1, and posts the wait object, wobj1. It then fills in the STATUS_LIST and return code rc3 for the WAIT_FOR_COMPLETION verb and returns control to TP(a). TP(a) is able to identify the status of the outstanding operations through the STATUS_LIST and their specific outcomes (e.g., OK or PROGRAM_STATE_CHECK) by checking the return codes corresponding to posted wait objects.

Examples Using Basic Conversation Verbs

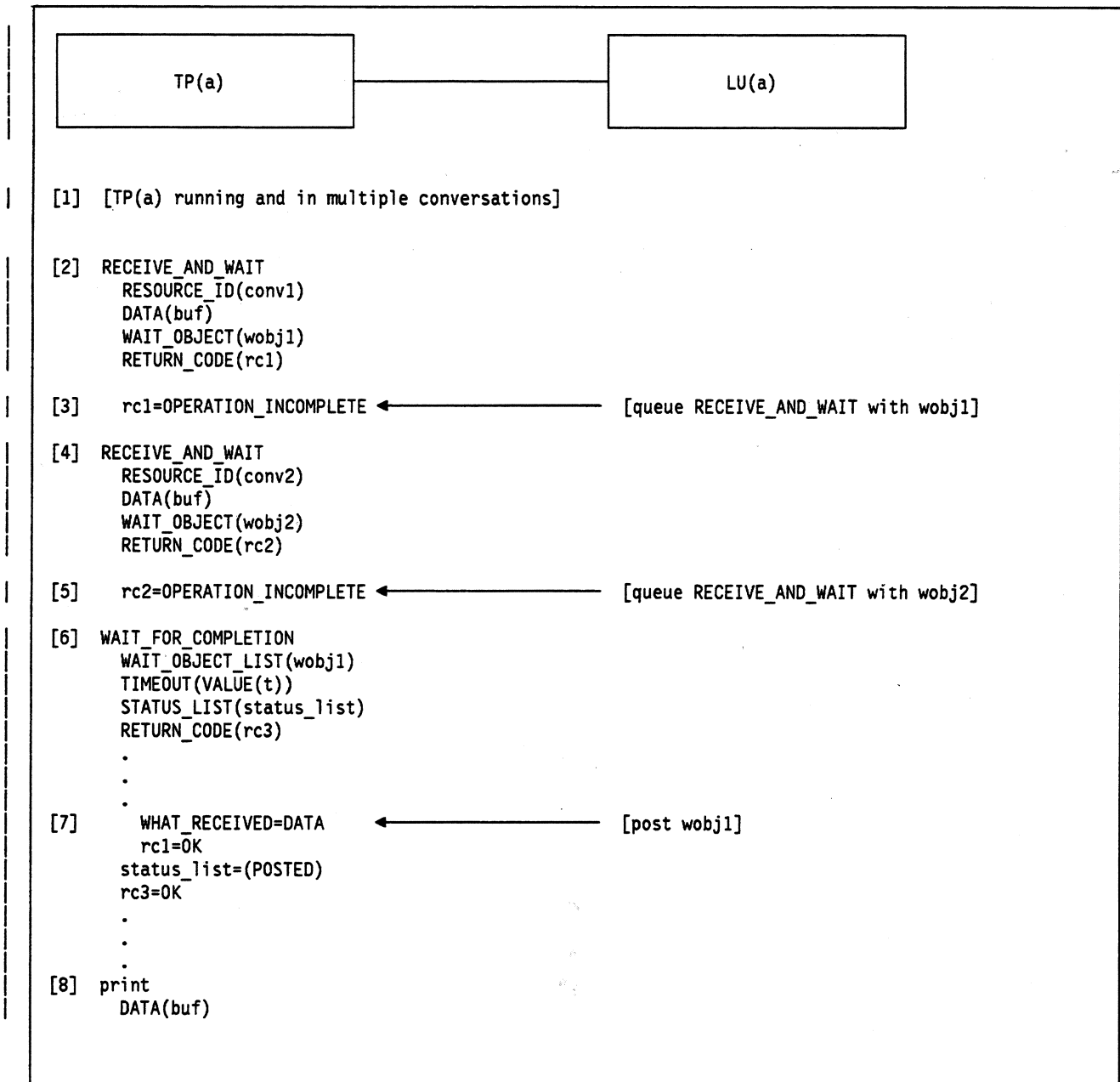


Figure B-15. Abnormal Reuse of Parameters in Nonblocking Verbs

Examples Using Basic Conversation Verbs

Notes for Figure B-15:

1. Assume TP(a) is in conversations with two partner TPs, and is now in receive state for both conversations.
2. TP(a) issues `RECEIVE_AND_WAIT` to its first conversation partner in a nonblocking manner, using buffer, buf.
3. LU(a) suspends the verb operation because of some resource shortage, and returns control to TP(a) with an indication of `OPERATION_INCOMPLETE`.
4. TP(a) subsequently issues another `RECEIVE_AND_WAIT` to its second conversation partner in a nonblocking manner, using the same buffer, buf, as it did in the previous request.
5. LU(a) suspends the verb operation again because of resource shortage, and returns

control to TP(a) with an indication of `OPERATION_INCOMPLETE`.

6. TP(a) monitors the status of the first outstanding operation by issuing `WAIT_FOR_COMPLETION`.
7. LU(a) completes the first request, fills the buffer, buf, sets the return code, rc1, and posts the wait object, wobj1. It then fills in the `STATUS_LIST` and return code rc3 for the `WAIT_FOR_COMPLETION` verb and returns control to TP(a). TP(a) is able to identify the completion status of the first outstanding request through the `STATUS_LIST` and its outcome by checking the return code rc1.
8. TP(a) proceeds to print the received information in the buffer, buf.

The printing output can be unpredictable because an outstanding verb operation is using the same buffer.

Examples Using Basic Conversation Verbs

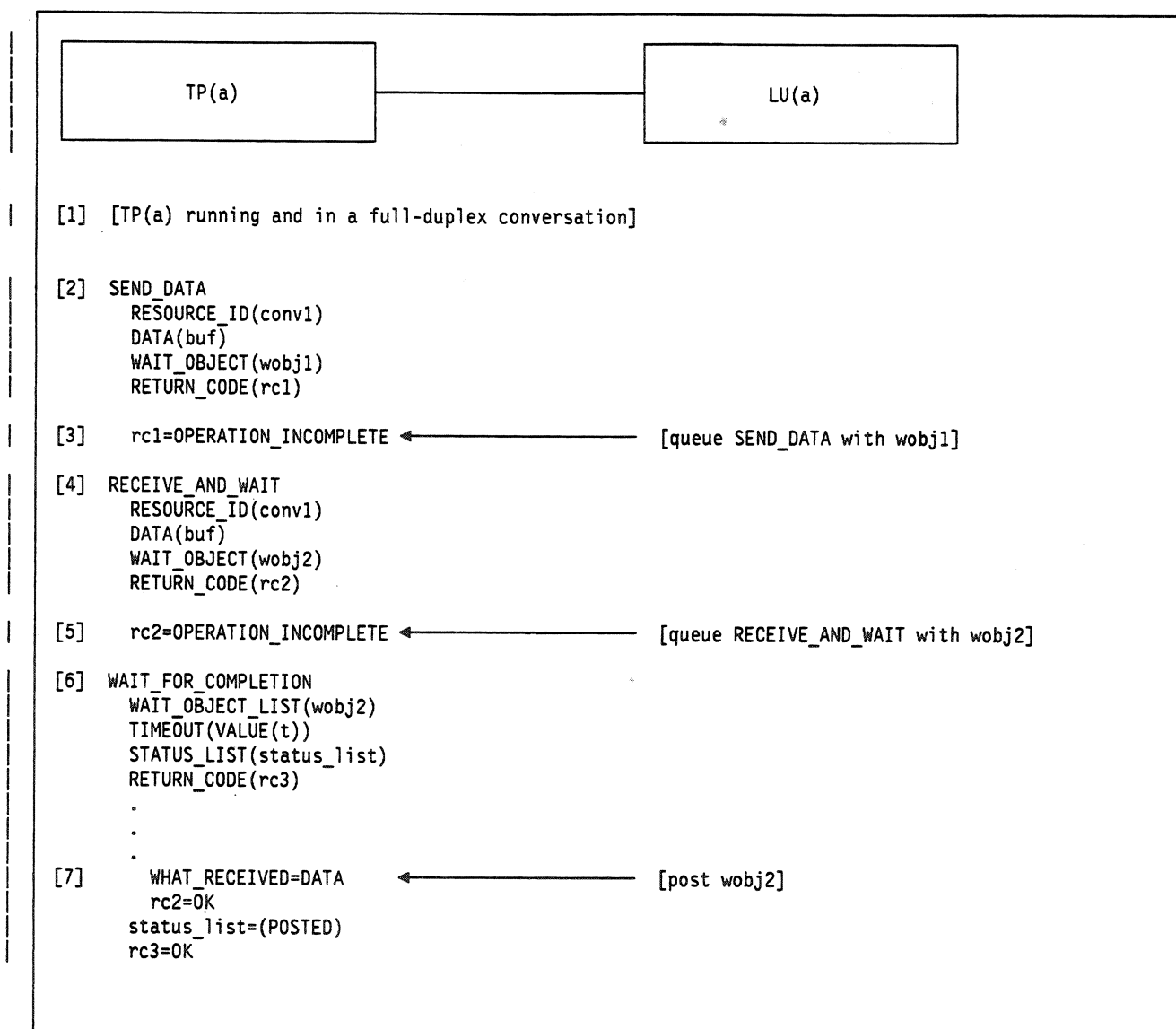


Figure B-16. Abnormal Reuse of Parameters in Nonblocking Verbs for Full-duplex Conversations Only

Notes for Figure B-16:

1. Assume TP(a) is in conversation with a partner TP, and is now in send-receive state.
2. TP(a) issues SEND_DATA to its partner in a nonblocking manner, using buffer, buf.
3. LU(a) suspends the verb operation because of some resource shortage, and returns control to TP(a) with an indication of OPERATION_INCOMPLETE.
4. TP(a) subsequently issues RECEIVE_AND_WAIT to the same partner in a nonblocking manner, using the same buffer, buf, as in the previous request.
5. LU(a) suspends the verb operation again because of resource shortage, and returns control to TP(a) with an indication of OPERATION_INCOMPLETE.
6. TP(a) monitors the status of the second outstanding operation by issuing WAIT_FOR_COMPLETION.
7. LU(a) completes the second request, fills the buffer, buf, sets the return code, rc2, and posts the wait object, wobj2. It then fills in the STATUS_LIST and return code rc3 for the WAIT_FOR_COMPLETION verb and returns control to TP(a). TP(a) identifies the completion status of the second outstanding request through the STATUS_LIST and its outcome by checking the return code rc2.

For a half-duplex conversation, TP(a) could conclude that it has successfully received the reply from its partner. For this full-duplex case, however, this is not necessarily true because the execution order of these two verbs, which belong to two different queues in the full-duplex conversation, is unpredictable. Therefore, the receipt of data into the shared buffer, buf, could mean the send operation will transmit faulty data.

Examples Using Basic Conversation Verbs

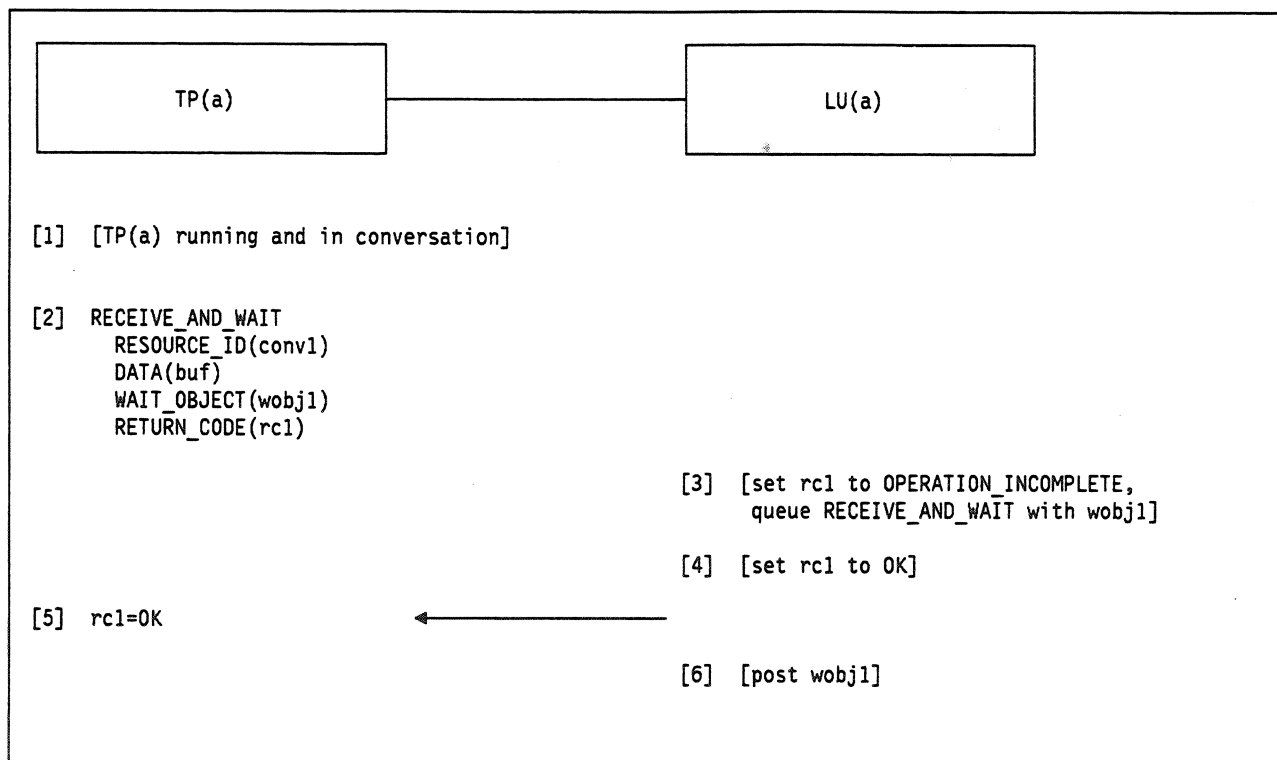


Figure B-17. Race Condition for Setting and Checking Nonblocking Verb Parameters

Notes for Figure B-17:

1. Assume TP(a) is in conversation with a partner TP, and is now in receive state.
2. TP(a) issues `RECEIVE_AND_WAIT` to its conversation partner in a nonblocking manner, using buffer, buf.
3. Recognizing a resource shortage, LU(a) suspends the verb operation, sets the `RETURN_CODE` to `OPERATION_INCOMPLETE`, and enables TP(a) to regain control.
4. However, by the time TP(a) regains control, LU(a) has been able to resume the outstanding operation, to complete the request, and to change the `RETURN_CODE` to `OK`.

5. Before LU(a) posts the wait object, wobj1, TP(a) regains control and identifies the `OK` return code. But the wait object, wobj1, has not been posted.
6. Finally, LU(a) proceeds and posts the wait object, wobj1.

Complications can arise in the following situations:

- TP(a) may find buf empty if the return code is not filled last by LU(a). This is why the properly implemented LU sets the return code after setting the other returned parameters.
- TP(a) reuses wobj1 between [5] and [6]. For this reason, the TP should not reuse a wait object until after it has verified its posting.

Examples Using Basic Conversation Verbs

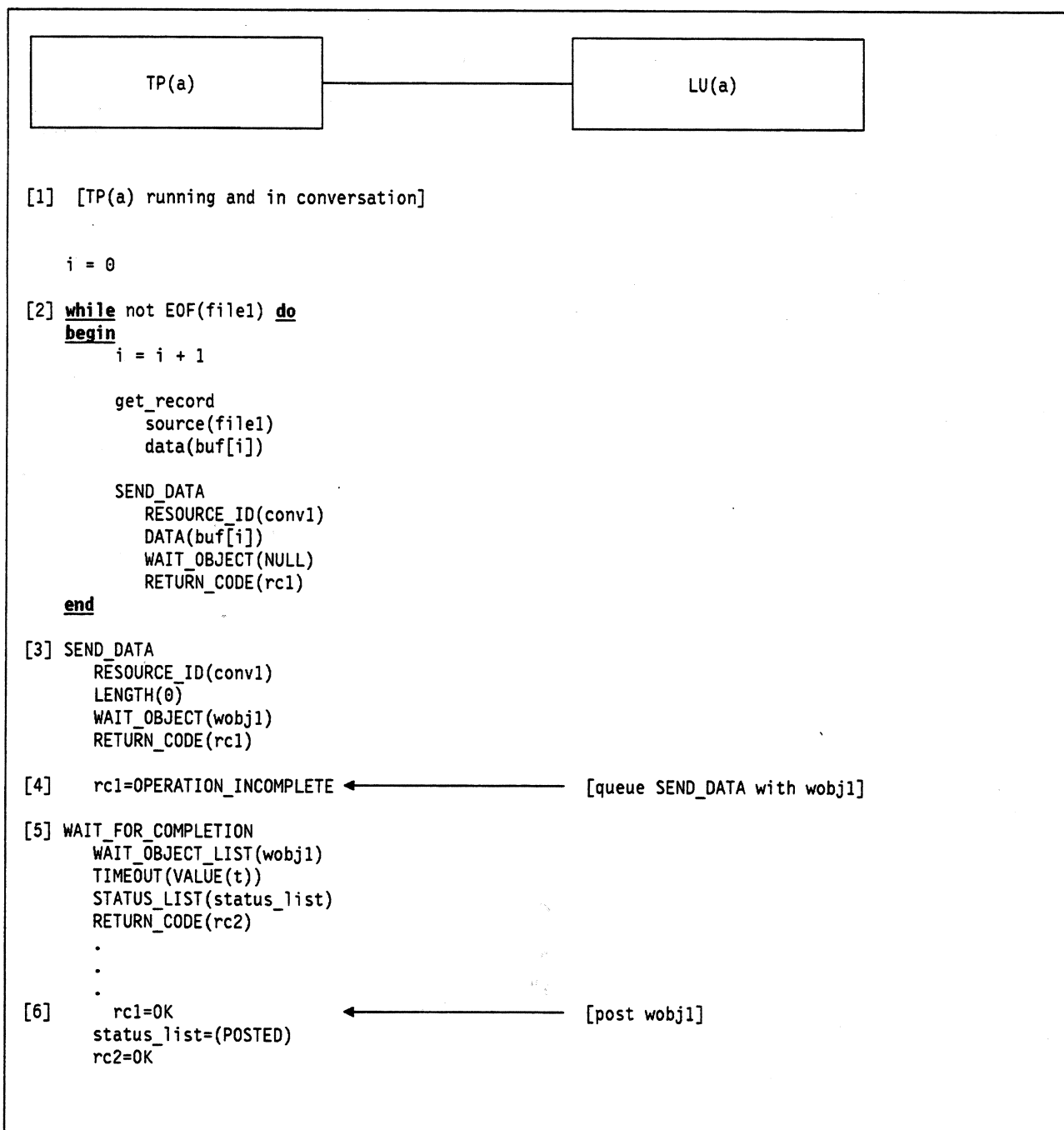


Figure B-18. NULL Wait Object

Notes for Figure B-18:

1. Assume TP(a) is in conversation with a partner TP, and is now in send state. TP(a) intends to transfer a large-sized file to its partner.
2. TP(a) is in a loop fetching data from the local store, file1, buffer by buffer and sending them to its partner via a nonblocking SEND_DATA with NULL wait object, indicating no confirmation is needed.
3. Upon having fetched all the data in file1, TP(a) issues another nonblocking SEND_DATA with no data and, this time, a wait object, wobj1.
4. Assume that LU(a) cannot complete the no-data request because of the size of file1. This is the case, since the incomplete verbs (still represented in the send queue) prevent the LU from processing the no-data send

request. LU(a) suspends the new verb operation and returns control to TP(a) with an indication of OPERATION_INCOMPLETE.

5. TP(a) monitors the status of the very last (no-data) send request by issuing WAIT_FOR_COMPLETION.
6. LU(a) completes the no-data send request, sets the return code, rc1, and posts the wait object, wobj1. It then fills in the STATUS_LIST and return code rc2 for the WAIT_FOR_COMPLETION verb and returns control to TP(a). TP(a) identifies the completion status of the last (no-data) outstanding operation through the STATUS_LIST and its outcome (e.g., OK or PROGRAM_STATE_CHECK) by checking the return code rc1.

TP(a) is now able to infer the completion of all the previous send requests, and, hence, the file transfer.

Examples Using Basic Conversation Verbs

Appendix C. Symbol String Conventions

This book refers to the following symbol strings:

- Network ID
- LU name
- Fully qualified LU name (synonymous with *network-qualified LU name*)
- Mode name
- Transaction program name
- SECURITY subfields
- PIP subfields
- Map name

This appendix defines the type and length of these symbol strings. The meanings of these symbol strings are defined in the chapters describing the individual verbs that refer to these symbol strings. The type and length of each symbol string is defined in terms of the send and receive support of all LU 6.2 products that implement the symbol string.

This appendix also discusses symbol string processing and conversion.

Symbol String Type

The symbol-string type identifies the set of characters from which the symbol string can be composed, and therefore the characters a transaction program can use to specify the symbol string. The following symbol-string types are defined:

- Type A (Assembler-oriented): a character string consisting of one or more EBCDIC uppercase letters A through Z; numerics 0 through 9; and special characters \$, #, and @; the first character of which is an uppercase letter or a special character.
- Type 1134 (Type A subset): a character string consisting of one or more EBCDIC uppercase letters A through Z; numerics 0 through 9; the first character of which can be any of these characters.

Notes:

1. The inclusion of symbol-string type 1134 into (and the exclusion of symbol-string type A from) IBM LU 6.2 product implementations will take place over time.
 2. IBM LU 6.2 product implementations use symbol-string type 1134 with the *restriction* that the first character of the symbol string must be an uppercase letter A through Z.
- Type AE (A-extended): a character string consisting of one or more EBCDIC lowercase letters a through z; uppercase letters A through Z; numerics 0 through 9; special characters \$, #, @; and the period (.); with no restriction on the first character.

Symbol Strings

- Type GR (EBCDIC graphics): a character string consisting of one or more EBCDIC characters in the range hex 41 through hex FE with no restriction on the first character.
- Type DB (double byte): a byte string consisting of an even number of four or more bytes beginning with a byte of hex 0E, followed by bytes in the range hex 41 through hex FE, and ending with a byte of hex 0F.
- Type G (general): a byte string consisting of one or more bytes in the range hex 00 through hex FF, with no restriction on the first byte.

The set of type-A and type-AE characters, and the hex codes for these characters, are shown in Figure C-1.

Hex Code	Graphic	Description	Set			Hex Code	Graphic	Description	Set		
			A	AE	1134				A	AE	1134
4B	.	Period		X		C4	D	D, Capital	X	X	X
5B	\$	Dollar Sign	X	X		C5	E	E, Capital	X	X	X
7B	#	Number Sign	X	X		C6	F	F, Capital	X	X	X
7C	@	At Sign	X	X		C7	G	G, Capital	X	X	X
81	a	a, Small		X		C8	H	H, Capital	X	X	X
82	b	b, Small		X		C9	I	I, Capital	X	X	X
83	c	c, Small		X		D1	J	J, Capital	X	X	X
84	d	d, Small		X		D2	K	K, Capital	X	X	X
85	e	e, Small		X		D3	L	L, Capital	X	X	X
86	f	f, Small		X		D4	M	M, Capital	X	X	X
87	g	g, Small		X		D5	N	N, Capital	X	X	X
88	h	h, Small		X		D6	O	O, Capital	X	X	X
89	i	i, Small		X		D7	P	P, Capital	X	X	X
91	j	j, Small		X		D8	Q	Q, Capital	X	X	X
92	k	k, Small		X		D9	R	R, Capital	X	X	X
93	l	l, Small		X		E2	S	S, Capital	X	X	X
94	m	m, Small		X		E3	T	T, Capital	X	X	X
95	n	n, Small		X		E4	U	U, Capital	X	X	X
96	o	o, Small		X		E5	V	V, Capital	X	X	X
97	p	p, Small		X		E6	W	W, Capital	X	X	X
98	q	q, Small		X		E7	X	X, Capital	X	X	X
99	r	r, Small		X		E8	Y	Y, Capital	X	X	X
A2	s	s, Small		X		E9	Z	Z, Capital	X	X	X
A3	t	t, Small		X		F0	0	Zero	X	X	X
A4	u	u, Small		X		F1	1	One	X	X	X
A5	v	v, Small		X		F2	2	Two	X	X	X
A6	w	w, Small		X		F3	3	Three	X	X	X
A7	x	x, Small		X		F4	4	Four	X	X	X
A8	y	y, Small		X		F5	5	Five	X	X	X
A9	z	z, Small		X		F6	6	Six	X	X	X
C1	A	A, Capital	X	X	X	F7	7	Seven	X	X	X
C2	B	B, Capital	X	X	X	F8	8	Eight	X	X	X
C3	C	C, Capital	X	X	X	F9	9	Nine	X	X	X

Figure C-1. Character Sets A and AE and 1134

Table C-1 on page C-3 defines the product send support and receive support for each symbol string in terms of the symbol-string types. Depending on the symbol string, product send support or receive support is indicated either by a single type or multiple types. Where multiple types are indicated, the type selected is product-defined and send support may differ from receive support.

Table C-1. Symbol-String Types		
Symbol String	Type	
	Send Support	Receive Support
Network ID	1134	1134
LU Name [1]	—	—
Fully Qualified LU Name [2]	1134	1134
Mode Name [5]	1134	1134
Transaction Program Name [3]	AE, GR, or DB	A, AE, GR, or DB
LU-LU Password [4]	—	—
SECURITY Subfields	AE, GR, DB, or 1134	A, AE, GR, DB or 1134
PIP Subfields	G	G
Map Name	A, AE, or GR	A, AE, or GR
Notes: 1. The LU name is a locally known name; it is the name by which one LU knows another LU. A transaction program specifies this LU name in conjunction with the mode name when it allocates a session for a conversation. This LU name is not sent outside the LU. The symbol-string type is G. 2. The fully qualified LU name consists of two symbol strings of type 1134 (type A in back-level implementations) concatenated by a period (.). The lefthand symbol string represents the network ID; the righthand symbol string represents the network LU name. The period is not part of the network ID or the network LU name. 3. The first character of an SNA service transaction program name is a character ranging in value from hex 00 through hex 0D and hex 10 through hex 3F (excluding hex 0E and hex 0F). More details about SNA service transaction program names and a list of SNA service transaction programs is given in Appendix D, "List of SNA Service Transaction Programs." 4. The LU-LU password is a locally-specified value and is not sent outside the LU. The symbol-string type is G. 5. All IBM-defined user session mode names begin with X'7B' as an indicator that this is IBM-defined. In various countries, X'7B' has various representations. In the United States, for example, X'7B' is the EBCDIC code point for #.		
A = Assembler, AE = Assembler-Extended, GR = Graphics, DB = Double Byte, G = General		

Symbol String Length

The symbol-string length represents the number of characters a symbol string can contain. Three symbol-string lengths are defined:

- Minimum specification length: the minimum number of characters that a transaction program is allowed to use to specify the symbol string. For some symbol strings, the minimum specification length is zero. Zero-length strings are valid symbol strings and are subject to the same usage conditions as valid non-zero length strings.¹
- Maximum send support: the maximum number of characters that all products can send for the symbol string.
- Maximum receive support: the maximum number of characters that all products can receive for the symbol string.

The maximum send or receive support for a symbol string's length is defined either by a single value or within a range of values, depending on the symbol string.

The single value is the maximum number of characters in a symbol string that all products can send or receive.

The range of values represents a lower and upper bound of the maximum number of characters in a symbol string that a product can send or receive. The specific maximum number of characters a product can send or receive for each of these symbol strings is product-defined within the range.

Table C-2 on page C-5 defines the product maximum send and receive support for each symbol string in terms of the symbol-string lengths. Where support is defined to be within a range of values, the range is given as "lower-value < — > upper-value," which identifies the lower and upper bounds of the range.

Note: The variable to which a type-A, type-AE, type-GR, or type-DB symbol string is assigned may be longer than the symbol string; in this case, the symbol string is left-justified within the variable and the variable is filled out to the right with space (hex 40) characters. Space characters, if present, are not part of the symbol string. If the symbol string is formed from the concatenation of two or more individual symbol strings, such as the fully qualified LU name, the concatenated symbol string as a whole is left-justified within the variable and the variable is filled out to the right with space characters. Space characters, if present, are not part of the concatenated symbol string.

¹ A *valid* symbol string is one that meets the requirements of the symbol-string type defined for that symbol string.

Table C-2. Symbol-String Lengths

Symbol String	Length		
	Minimum Specification	Maximum Send Support	Maximum Receive Support
Network ID	0	8	8
LU Name [1]	0	—	—
Fully Qualified LU Name	1	17	17
Mode Name	0	8	8
Transaction Program Name	1	8 <—> 64	8 <—> 64
LU-LU Password [2]	1	—	—
SECURITY Subfields	0	8 <—> 10	0 <—> 10
PIP Subfields [3]	0	64 <—> * [4]	64 <—> * [5]
Map Name	1	8 <—> 64	7 <—> 64

Notes:

1. The LU name is a locally known name; it is the name by which one LU knows another LU. A transaction program specifies this LU name in conjunction with the mode name when it allocates a session for a conversation. This LU name is not sent outside the LU. The maximum specification length of the LU name is product-defined.
2. The LU-LU password is a locally-specified value and is not sent outside the LU. It is 64 bits (8 bytes) in length. At least 8 bits (1 byte) must be specified. If less than 64 bits are specified, the local LU fills the password out to the right with binary 0's.
3. Product support of PIP subfields is optional, and send support is independent of receive support. The maximum number of PIP subfields a product can send or receive is product-defined; it can be any number greater than or equal to 7.
4. The maximum send support for PIP subfields is product-defined; it can be any length greater than or equal to 64.
5. The maximum receive support for PIP subfields is product-defined; it can be any length greater than or equal to 64.

Symbol String Processing Conventions for Symbol Strings A, 1134, AE, GR, and DB

No symbol string processing rules are defined by LU 6.2. The transaction programs and the products implementing LU 6.2, however, may have conventions and rules of their own.

Symbol Strings

Uppercase and Lowercase

Uppercase and lowercase characters may be intermixed. Lowercase characters are considered identical to their corresponding uppercase characters. The bit patterns, however, do not need to be changed to guarantee equivalent processing (i.e., no mapping is necessary from lowercase to uppercase characters but such might be done).

Leading and Trailing Blanks

LU 6.2 disallows leading and trailing blanks for all symbol strings of types A, AE, GR, and DB.

Imbedded Blanks

LU 6.2 disallows imbedded blanks for all symbol strings of types A, AE, GR, and DB.

Imbedded Special Characters

LU 6.2 disallows imbedded special characters only when the special characters are not defined within the symbol string type for a symbol string.

Invalid Characters

LU 6.2 disallows any symbol string that contains one or more characters not defined as being within the symbol string type for a symbol string.

Symbol String Processing with Symbol String G

No symbol string processing rules are defined by LU 6.2. The transaction programs and the products implementing LU 6.2, however, may have conventions and rules of their own.

Uppercase and Lowercase

Uppercase and lowercase characters may be intermixed. Lowercase characters are considered identical to their corresponding uppercase characters. The bit patterns, however, do not need to be changed to guarantee equivalent processing (i.e., no mapping is necessary from lowercase to uppercase characters but such might be done).

Leading and Trailing Blanks

LU 6.2 disallows leading and trailing blanks for all symbol strings of type G even though the blank character (X'40') is included in symbol string type G.

Imbedded Blanks

LU 6.2 disallows imbedded blanks for all symbol strings of type G even though the blank character (X'40') is included in symbol string type G (general).

Imbedded Special Characters

LU 6.2 disallows imbedded special characters only when the special characters are not defined within the symbol string type for a symbol string.

Invalid Characters

LU 6.2 disallows any symbol string that contains one or more characters not defined as being within the symbol string type for a symbol string.

Symbol String Conversions

There are no symbol string conversions defined by LU 6.2. Any symbol string conversions are transparent to LU 6.2.

Symbol Strings

Appendix D. List of SNA Service Transaction Programs

This appendix lists the classes of SNA service transaction programs. The SNA service transaction programs are categorized according to functional classes. A class is identified by the first one or two characters of the name. All SNA service transaction programs belonging to a given class have the same class identifier.

The SNA service transaction program classes are:

Class	Identifier (in hex)
Scheduler	02
Queue	03
DL/1	05
Change Number of Sessions	06F1
Resynchronization	06F2
Security	06F3
Distributed Data Management	07F0
Distr. Relational Database Arch.	07F6
Document Interchange Architecture	20F0
SNA Distribution Services	21F0
Advanced Peer-to-Peer Networking	22F0
Management Services	23F0
File Services	24F0
Product-Oriented	30F0

SNA Service Transaction Program Names

SNA service transaction programs are distinguished by their names, in particular the first (leftmost) character. The name of an SNA service transaction program can be one to four characters in length; typically, however, they are four characters in length. The first character of the name can range in value from hex 00 through hex 0D and hex 10 through hex 3F (excluding hex 0E and hex 0F). The remaining characters of the name are type-A, without any restriction on the first type-A character. By contrast, names of programs other than SNA service transaction programs are type-A, type-AE, type-GR, or type-DB symbol strings.

Using the first character of the name, a product may restrict the right of access of its programs to SNA service transaction programs. For example, a product may allow only its "privileged" programs to allocate conversations to SNA service transaction programs, and permit its "nonprivileged" programs to allocate conversations only to transaction programs other than SNA service transaction programs.

Listed below are the individual SNA service transaction programs, grouped by class.

SNA Service Transaction Programs

Scheduler

Name (in hex)	Description
02	LU 6.1 scheduler transaction program

See the supporting IBM product's publications for more details.

Queue

Name (in hex)	Description
03	LU 6.1 queue transaction program

See the supporting IBM product's publications for more details.

DL/1

Name (in hex)	Description
05	LU 6.1 DL/1 transaction program

See the supporting IBM product's publications for more details.

Change Number of Sessions

Name (in hex)	Description
06F1	CNOS service transaction program

See *SNA LU 6.2 Reference: Peer Protocols* for more details.

Resynchronization

Name (in hex)	Description
06F2	Sync point resynchronization transaction program
06F2F0F1	Sync point message transaction program

See *SNA LU 6.2 Reference: Peer Protocols* and the supporting IBM product's publications for more details.

Security

Name (in hex)	Description
06F3F0F0	Sign-Off transaction program
06F3F0F1	Sign-On/Change-Password transaction program

See *SNA LU 6.2 Reference: Peer Protocols* for more details.

Distributed Data Management

Name (in hex)	Description
07F0F0F1	Distributed data management synchronous conversation transaction program
07F0F0F2	Distributed data management simplex conversation transaction program

See the supporting IBM product's publications for more details.

Distributed Relational Database Architecture

Name (in hex)	Description
07F6C4C2	Distributed Relational Database Architecture Default Application Server transaction program

See the *Distributed Relational Database Architecture Reference* and the supporting IBM product publications for more details.

Document Interchange Architecture

Name (in hex)	Description
20F0F0F0	DIA process transaction program
20F0F0F1	DIA file server transaction program
20F0F0F2	DIA Status Receiver transaction program
20F0F0F3	DIA TP name for SNADS
20F0F0F4	DIA server name for SNADS
20F0F0F5	DIA Services — SNADS DIA Agent
20F0F0F6	DIA Services — SNADS DIA Server
20F0F0F7	DIA Services — DIA General Delivery Service
20F0F0F8	DIA Services — DIA Library Service
20F0F0F9	DIA Services — DIA Application Service

See *Document Interchange Architecture: Technical Reference* for more details.

SNA Distribution Services

Name (in hex)	Description
21F0F0F1	SNADS DS_SEND transaction program
21F0F0F2	SNADS DS_RECEIVE transaction program
21F0F0F3	SNADS DS_ROUTER_DIRECTOR transaction program
21F0F0F6	SNADS general server transaction program
21F0F0F7	Release 2-Encoding of SNADS DS_SEND transaction program
21F0F0F8	Release 2-Encoding of SNADS DS_RECEIVE transaction program

See *SNA Format and Protocol Reference Manual: Distribution Services* for more details.

APPN

Name (in hex)	Description
22F0F0F1	CP_CAP_SON transaction program
22F0F0F2	RESOURCE_REGISTRATION transaction program
22F0F0F3	RECEIVE_NETWORK_SEARCH transaction program
22F0F0F4	TOPOLOGY_DATABASE_UPDATE transaction program

See *SNA APPN Architecture Reference* for more details.

SNA Service Transaction Programs

Management Services

Name (in hex)	Description
23F0F0F0	MSU_HANDLER transaction program
23F0F0F1	MDS_RECEIVE transaction program
23F0F0F2	MDS_HP_RECEIVE transaction program

See *SNA Management Services Reference* for more details.

File Services

Name (in hex)	Description
24F0F0F0	FS Server transaction program

See *SNA File Services Reference* for more details.

Product-Oriented

The following SNA service transaction programs are provided by specific products.

Name (in hex)	Description
30F0F0F0	Printer CPDS transaction program for IBM 3820
30F0F0F1	Printer level 2 transaction program for IBM 3820
30F0F0F2	Object distribution transaction program for IBM System/38
30F0F0F3	Object distribution server transaction program for IBM System/38
30F0F0F4	Transfer Facility for S/36
30F0F0F5	IBM 5250 device passthrough transaction program for IBM System/36 and IBM System/38
30F0F0F6	Virtual disk transaction program for IBM System/36 and IBM System/38
30F0F0F7	Virtual printer transaction program for IBM System/36 and IBM System/38
30F0F0F8	PC Support/36 Shared Folder/File Support
30F0F1F5	AS/400, Virtual Print
30F0F1F6	AS/400, Transfer Facility
30F0F1F7	AS/400, Message Facility 1
30F0F1F8	AS/400, Message Facility 2
30F0F1F9	AS/400, Shared Folders
30F0F4F5	AS/400, 3270 Passthrough for LU2-to/from-LU6.2
30F0F4F6	AS/400, PC Support for Shared Folders
30F0F5F4	AS/400, Systems Management Utilities - Focal Point Support

See the appropriate IBM product's publications for more details.

Appendix E. Conversation State Matrices

Conversation State Matrices

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

The state of a conversation is defined in terms of the local program's view of the local end of the conversation. The local end of the conversation is the end to which the local program is connected. The states of other conversations allocated to the program can be different. For example, one conversation can be in receive state and another in send state, concurrently.

This appendix shows the conversation state transitions that can occur when a program issues a conversation verb. A state transition can occur as a result of a verb the local program issued, a verb the remote program issued, or a network error; the latter two are indicated when a return code of other than OK is returned to the local program.

The conversation state transitions are represented by means of a matrix; see Figure E-1, through Figure E-3, and Figure E-7. The columns of the matrices show the individual states, and the rows show the individual verbs. A verb is shown more than once when a parameter of the verb or the return code determines the state transitions that can occur.

The state of the conversation determines the verbs that a program is allowed to issue. Following the state-transition matrices are matrices showing the state-check conditions that can occur; see Figure E-5 and Figure E-8. A state check occurs when the program attempts to issue a verb for a conversation that is in a state in which the verb is not allowed. The individual verb descriptions list the applicable state-check conditions.

The SYNCPT and BACKOUT verbs apply to all protected resources at the time the verbs are issued, and only to protected resources. A conversation is protected when it is allocated with a synchronization level of SYNCPT. Therefore, the correlation of the SYNCPT and BACKOUT verbs to conversation states applies only to protected conversations. The states of unprotected conversations — those allocated with a synchronization level other than SYNCPT — are not affected by the SYNCPT and BACKOUT verbs.

For conversations allocated with SYNC_LEVEL(SYNCPT), the backout-required state can be entered on some conversations because of events occurring on a different conversation, i.e., a return code of BACKED_OUT, RESOURCE_FAILURE_*, or DEALLOCATE_ABEND*.

Conversation State Matrices

Abbreviations are used for the parameters, return codes, and what-received indications. The abbreviations and symbols used in the state-transition matrix are defined in Figure E-4. Abbreviations and symbols used in the state-check matrix are defined at the bottom of Figure E-5.

Half-Duplex Conversation State Matrices

The following half-duplex conversation states are defined at the conversation protocol boundary (where the prefix [MC_] in a verb name means the verb can be either a mapped or basic conversation verb):

Reset — the state in which the program can allocate the conversation.

Send — the state in which the program can send data, request confirmation, or request sync point.

Defer Receive and Defer Deallocate — the states in which the program can request sync point services. These states are only entered when the synchronization level is SYNCPT. In the Defer Receive state, the program can also request confirmation or simply flush the LU's send buffer. The Defer states affect the state following a subsequent successful SYNCPT verb: the resulting state is Receive from the Defer Receive state and Reset from the Defer Deallocate state.

Receive — the state in which the program can receive information from the remote program.

Confirm, Confirm Send, and Confirm Deallocate — the states in which the program can reply to a confirmation request. The Confirm states affect the state following a successful CONFIRMED verb: the resulting state is Receive from the Confirm state, Send from the Confirm Send state, and Reset from the Confirm Deallocate state.

Sync Point, Sync Point Send, and Sync Point Deallocate — the states in which the program can respond to a sync point request. These states are entered when the program receives the WHAT_RECEIVED indicator of TAKE_SYNCPT, TAKE_SYNCPT_SEND, or TAKE_SYNCPT_DEALLOCATE, or when it issues the (MC_)PREPARE_FOR_SYNCPT verb. The sync point states affect the state following a successful SYNCPT verb: the resulting state is Receive from the Sync Point state, Send from the Sync Point Send state, and Reset from the Sync Point Deallocate state.

Deallocate — the state in which the program can deallocate the conversation locally.

Deallocate Pending — the state in which the program cannot exchange any expedited data. This state is entered when processing of a DEALLOCATE TYPE(CONFIRM) verb is suspended and awaits for a reply to the CONFIRM request from the partner TP.

Backout required — the state in which the program must issue the BACKOUT verb to propagate a backout caused by an event on this conversation or another conversation, such as a conversation failure, DEALLOCATE TYPE(ABEND), or BACKED_OUT return code on a conversation verb. These events will cause all protected conversations (those allocated with SYNC_LEVEL(SYNCPT)) that are not in Reset or Deallocate state to enter the Backout Required state. The BACKOUT verb will cause all these conversations to exit the Backout Required state. In this state, the program may issue DEALLOCATE TYPE(ABEND*) against a specific conversation, but it still must issue BACKOUT before it can issue other verbs against other protected conversations or allocate new protected conversations. This state only occurs on conversations allocated with SYNC_LEVEL(SYNCPT).

Conversation State Matrices

Verb	Conversation States						
	Reset	Send	Defer Receive	Defer Deallo- cate	Receive	Confirm	Confirm Send
	1	2	3	4	5	6	7
Initiating Conversation	5	/	/	/	/	/	/
ALLOCATE[ok]	2	/	/	/	/	/	/
ALLOCATE[ae]	12	/	/	/	/	/	/
ALLOCATE[pe]	-	/	/	/	/	/	/
ALLOCATE[un]	-	/	/	/	/	/	/
BACKOUT	/	*	*	*	*	*	*
CONFIRM[ok]	/	-	5	/	/	/	/
CONFIRM[ae]	/	12	12	/	/	/	/
CONFIRM[bo]	/	13	13	/	/	/	/
CONFIRM[da]	/	12 ¹	12 ¹	/	/	/	/
CONFIRM[ep]	/	5	5	/	/	/	/
CONFIRM[me]	/	-	2	/	/	/	/
CONFIRM[rf]	/	12 ¹	12 ¹	/	/	/	/
CONFIRMED	/	/	/	/	/	5	2
DEALLOCATE(F)[ok]	/	1	/	/	/	/	/
DEALLOCATE(C)[ok]	/	1	/	/	/	/	/
DEALLOCATE(S)[ok]	/	4	/	/	/	/	/
DEALLOCATE(A)[ok]	/	1 ¹	1 ¹	1 ¹	1 ¹	1 ¹	1 ¹
DEALLOCATE(L)[ok]	/	/	/	/	/	/	/
DEALLOCATE(C)[ae]	/	12	/	/	/	/	/
DEALLOCATE(C)[da]	/	12	/	/	/	/	/
DEALLOCATE(C)[ep]	/	5	/	/	/	/	/
DEALLOCATE(C)[me]	/	-	/	/	/	/	/
DEALLOCATE(C)[oi] ²	/	14	/	/	/	/	/
DEALLOCATE(C)[rf]	/	12	/	/	/	/	/
FLUSH	/	-	5	/	/	/	/
GET_ATTRIBUTES	/	-	-	-	-	-	-
GET_TP_PROPERTIES	-	-	-	-	-	-	-
GET_TYPE	/	-	-	-	-	-	-
POST_ON_RECEIPT	/	/	/	/	-	/	/
PREPARE_FOR_SYNCPT[ok]	/	10	9	11	/	/	/
PREPARE_FOR_SYNCPT[ae]	/	12	12	12	/	/	/
PREPARE_FOR_SYNCPT[bo]	/	13	13	13	/	/	/
PREPARE_FOR_SYNCPT[da]	/	12 ¹	12 ¹	12 ¹	/	/	/
PREPARE_FOR_SYNCPT[en]	/	5	5	5	/	/	/
PREPARE_FOR_SYNCPT[ep]	/	5	5	5	/	/	/
PREPARE_FOR_SYNCPT[me]	/	-	/	/	/	/	/
PREPARE_FOR_SYNCPT[rf]	/	12 ¹	12 ¹	12 ¹	/	/	/
PREPARE_TO_RECEIVE[bo]	/	13	/	/	/	/	/
PREPARE_TO_RECEIVE(F)[ok]	/	5	/	/	/	/	/
PREPARE_TO_RECEIVE(C)[ok]	/	5	/	/	/	/	/
PREPARE_TO_RECEIVE(S)[ok]	/	3	/	/	/	/	/
PREPARE_TO_RECEIVE(C)[ae]	/	12	/	/	/	/	/
PREPARE_TO_RECEIVE(C)[da]	/	12 ¹	/	/	/	/	/
PREPARE_TO_RECEIVE(C)[ep]	/	5	/	/	/	/	/
PREPARE_TO_RECEIVE(C)[me]	/	-	/	/	/	/	/
PREPARE_TO_RECEIVE(C)[rf]	/	12 ¹	/	/	/	/	/

Figure E-1. Half-duplex Conversation State Transition Matrix (Part 1 of 3)

Conversation State Matrices

Conversation States (continued)							Verb
Confirm Deallocate 8	Sync-point 9	Sync-point Send 10	Sync-point Deallocate 11	Deallocate 12	Backout Required 13	Deallocate Pending 14	
/	/	/	/	/	/	/	Initiating Conversation
/	/	/	/	/	/	/	ALLOCATE[ok]
/	/	/	/	/	/	/	ALLOCATE[ae]
/	/	/	/	/	/	/	ALLOCATE[pe]
/	/	/	/	/	/	/	ALLOCATE[un]
*	*	*	*	-	*	*	BACKOUT
/	/	/	/	/	/	/	CONFIRM[ok]
/	/	/	/	/	/	/	CONFIRM[ae]
/	/	/	/	/	/	/	CONFIRM[bo]
/	/	/	/	/	/	/	CONFIRM[da]
/	/	/	/	/	/	/	CONFIRM[ep]
/	/	/	/	/	/	/	CONFIRM[me]
/	/	/	/	/	/	/	CONFIRM[rf]
12	/	/	/	/	/	/	CONFIRMED
/	/	/	/	/	/	/	DEALLOCATE(F)[ok]
/	/	/	/	/	/	1	DEALLOCATE(C)[ok]
/	/	/	/	/	/	/	DEALLOCATE(S)[ok]
1 ¹	1 ¹	1 ¹	1 ¹	/	/	/	DEALLOCATE(A)[ok]
/	/	/	/	1	/	/	DEALLOCATE(L)[ok]
/	/	/	/	/	/	/	DEALLOCATE(C)[ae]
/	/	/	/	/	/	/	DEALLOCATE(C)[da]
/	/	/	/	/	/	5	DEALLOCATE(C)[ep]
/	/	/	/	/	/	/	DEALLOCATE(C)[me]
/	/	/	/	/	/	/	DEALLOCATE(C)[oi] ²
/	/	/	/	/	/	/	DEALLOCATE(C)[rf]
/	/	/	/	/	/	/	FLUSH
-	-	-	-	-	-	-	GET_ATTRIBUTES
-	-	-	-	-	-	-	GET_TP_PROPERTIES
-	-	-	-	-	-	-	GET_TYPE
/	/	/	/	/	/	/	POST_ON_RECEIPT
/	/	/	/	/	/	/	PREPARE_FOR_SYNCPT[ok]
/	/	/	/	/	/	/	PREPARE_FOR_SYNCPT[ae]
/	/	/	/	/	/	/	PREPARE_FOR_SYNCPT[bo]
/	/	/	/	/	/	/	PREPARE_FOR_SYNCPT[da]
/	/	/	/	/	/	/	PREPARE_FOR_SYNCPT[en]
/	/	/	/	/	/	/	PREPARE_FOR_SYNCPT[ep]
/	/	/	/	/	/	/	PREPARE_FOR_SYNCPT[me]
/	/	/	/	/	/	/	PREPARE_FOR_SYNCPT[rf]
/	/	/	/	/	/	/	PREPARE_TO_RECEIVE[bo]
/	/	/	/	/	/	/	PREPARE_TO_RECEIVE(C)[ok]
/	/	/	/	/	/	/	PREPARE_TO_RECEIVE(S)[ok]
/	/	/	/	/	/	/	PREPARE_TO_RECEIVE(C)[ok]
/	/	/	/	/	/	/	PREPARE_TO_RECEIVE(F)[ae]
/	/	/	/	/	/	/	PREPARE_TO_RECEIVE(C)[da]
/	/	/	/	/	/	/	PREPARE_TO_RECEIVE(S)[ep]
/	/	/	/	/	/	/	PREPARE_TO_RECEIVE(S)[me]
/	/	/	/	/	/	/	PREPARE_TO_RECEIVE(C)[rf]

¹ Other protected conversations of the local program may be in the backout required state following this verb. Check (MC_)GET_ATTRIBUTES CONVERSATION_STATE to find out if this is so.

² OPERATION INCOMPLETE is resulted from the situation where processing of DEALLOCATE(C) was suspended because of waiting for a reply to the CONFIRM request from the partner TP.

Conversation State Matrices

Verb	Conversation States						
	Reset	Send	Defer Receive	Defer Deallo- cate	Receive	Confirm	Confirm Send
	1	2	3	4	5	6	7
RECEIVE_AND_WAIT[ok]{da}	/	5	/	/	-	/	/
RECEIVE_AND_WAIT[ok]{se}	/	-	/	/	2	/	/
RECEIVE_AND_WAIT[ok]{co}	/	6	/	/	6	/	/
RECEIVE_AND_WAIT[ok]{cs}	/	7	/	/	7	/	/
RECEIVE_AND_WAIT[ok]{cd}	/	8	/	/	8	/	/
RECEIVE_AND_WAIT[ok]{sy}	/	9	/	/	9	/	/
RECEIVE_AND_WAIT[ok]{ss}	/	10	/	/	10	/	/
RECEIVE_AND_WAIT[ok]{sd}	/	11	/	/	11	/	/
RECEIVE_AND_WAIT[ae]	/	12	/	/	12	/	/
RECEIVE_AND_WAIT[bo]	/	13	/	/	13	/	/
RECEIVE_AND_WAIT[da]	/	12 ¹	/	/	12 ¹	/	/
RECEIVE_AND_WAIT[dn]	/	12	/	/	12	/	/
RECEIVE_AND_WAIT[en]	/	5	/	/	-	/	/
RECEIVE_AND_WAIT[ep]	/	5	/	/	-	/	/
RECEIVE_AND_WAIT[et]	/	/	/	/	-	/	/
RECEIVE_AND_WAIT[me]	/	/	/	/	2	/	/
RECEIVE_AND_WAIT[rf]	/	12 ¹	/	/	12 ¹	/	/
RECEIVE_EXPEDITED_DATA	/	-	-	-	-	-	-
RECEIVE_IMMEDIATE[ok]{da}	/	/	/	/	-	/	/
RECEIVE_IMMEDIATE[ok]{se}	/	/	/	/	2	/	/
RECEIVE_IMMEDIATE[ok]{co}	/	/	/	/	6	/	/
RECEIVE_IMMEDIATE[ok]{cs}	/	/	/	/	7	/	/
RECEIVE_IMMEDIATE[ok]{cd}	/	/	/	/	8	/	/
RECEIVE_IMMEDIATE[ok]{sy}	/	/	/	/	9	/	/
RECEIVE_IMMEDIATE[ok]{ss}	/	/	/	/	10	/	/
RECEIVE_IMMEDIATE[ok]{sd}	/	/	/	/	11	/	/
RECEIVE_IMMEDIATE[ae]	/	/	/	/	12	/	/
RECEIVE_IMMEDIATE[bo]	/	/	/	/	*	/	/
RECEIVE_IMMEDIATE[da]	/	/	/	/	12 ¹	/	/
RECEIVE_IMMEDIATE[dn]	/	/	/	/	12	/	/
RECEIVE_IMMEDIATE[en]	/	/	/	/	-	/	/
RECEIVE_IMMEDIATE[ep]	/	/	/	/	-	/	/
RECEIVE_IMMEDIATE[et]	/	/	/	/	-	/	/
RECEIVE_IMMEDIATE[me]	/	/	/	/	2	/	/
RECEIVE_IMMEDIATE[rf]	/	/	/	/	12 ¹	/	/
RECEIVE_IMMEDIATE[un]	/	/	/	/	-	/	/
REQUEST_TO_SEND	/	-	/	/	-	-	/

Figure E-2. Half-duplex Conversation State Transition Matrix (Part 2 of 3)

Conversation State Matrices

Conversation States (continued)							Verb
Confirm Deallo- cate 8	Sync- point 9	Sync- point Send 10	Sync- point Deallo- cate 11	Deallo- cate 12	Backout Requi- red 13	Deallo- cate Pending 14	
/	/	/	/	/	/	/	RECEIVE_AND_WAIT[ok]{da}
/	/	/	/	/	/	/	RECEIVE_AND_WAIT[ok]{se}
/	/	/	/	/	/	/	RECEIVE_AND_WAIT[ok]{co}
/	/	/	/	/	/	/	RECEIVE_AND_WAIT[ok]{cs}
/	/	/	/	/	/	/	RECEIVE_AND_WAIT[ok]{cd}
/	/	/	/	/	/	/	RECEIVE_AND_WAIT[ok]{sy}
/	/	/	/	/	/	/	RECEIVE_AND_WAIT[ok]{ss}
/	/	/	/	/	/	/	RECEIVE_AND_WAIT[ok]{sd}
/	/	/	/	/	/	/	RECEIVE_AND_WAIT[ae]
/	/	/	/	/	/	/	RECEIVE_AND_WAIT[bo]
/	/	/	/	/	/	/	RECEIVE_AND_WAIT[da]
/	/	/	/	/	/	/	RECEIVE_AND_WAIT[dn]
/	/	/	/	/	/	/	RECEIVE_AND_WAIT[en]
/	/	/	/	/	/	/	RECEIVE_AND_WAIT[ep]
/	/	/	/	/	/	/	RECEIVE_AND_WAIT[et]
/	/	/	/	/	/	/	RECEIVE_AND_WAIT[me]
/	/	/	/	/	/	/	RECEIVE_AND_WAIT[rf]
-	-	-	-	/	-	/	RECEIVE_EXPEDITED DATA
/	/	/	/	/	/	/	RECEIVE_IMMEDIATE[ok]{da}
/	/	/	/	/	/	/	RECEIVE_IMMEDIATE[ok]{se}
/	/	/	/	/	/	/	RECEIVE_IMMEDIATE[ok]{co}
/	/	/	/	/	/	/	RECEIVE_IMMEDIATE[ok]{cs}
/	/	/	/	/	/	/	RECEIVE_IMMEDIATE[ok]{cd}
/	/	/	/	/	/	/	RECEIVE_IMMEDIATE[ok]{sy}
/	/	/	/	/	/	/	RECEIVE_IMMEDIATE[ok]{ss}
/	/	/	/	/	/	/	RECEIVE_IMMEDIATE[ok]{sd}
/	/	/	/	/	/	/	RECEIVE_IMMEDIATE[ae]
/	/	/	/	/	/	/	RECEIVE_IMMEDIATE[bo]
/	/	/	/	/	/	/	RECEIVE_IMMEDIATE[da]
/	/	/	/	/	/	/	RECEIVE_IMMEDIATE[dn]
/	/	/	/	/	/	/	RECEIVE_IMMEDIATE[en]
/	/	/	/	/	/	/	RECEIVE_IMMEDIATE[ep]
/	/	/	/	/	/	/	RECEIVE_IMMEDIATE[et]
/	/	/	/	/	/	/	RECEIVE_IMMEDIATE[me]
/	/	/	/	/	/	/	RECEIVE_IMMEDIATE[rf]
/	/	/	/	/	/	/	RECEIVE_IMMEDIATE[un]
-	-	-	-	/	/	/	REQUEST_TO_SEND

¹ Other protected conversations of the local program may be in the backout required state following this verb. Check (MC_)GET_ATTRIBUTES CONVERSATION_STATE to find out if this is so.

Conversation State Matrices

Verb	Conversation States						
	Reset	Send	Defer Receive	Defer Deallo- cate	Receive	Confirm	Confirm Send
	1	2	3	4	5	6	7
SEND_DATA[ok]	/	-	/	/	/	/	/
SEND_DATA[ae]	/	12	/	/	/	/	/
SEND_DATA[bo]	/	13	/	/	/	/	/
SEND_DATA[da]	/	12 ¹	/	/	/	/	/
SEND_DATA[ep]	/	5	/	/	/	/	/
SEND_DATA[me]	/	-	/	/	/	/	/
SEND_DATA[rf]	/	12 ¹	/	/	/	/	/
SEND_ERROR[ok] ²	/	-	/	/	2	2	2
SEND_ERROR[ok] ³	/	-	/	/	2	2	2
SEND_ERROR[ae]	/	12	/	/	/	/	/
SEND_ERROR[bo]	/	13	/	/	/	/	/
SEND_ERROR[da]	/	12 ¹	/	/	/	/	/
SEND_ERROR[dn]	/	/	/	/	12 ¹	/	/
SEND_ERROR[ep]	/	5	/	/	/	/	/
SEND_ERROR[me]	/	-	/	/	/	/	/
SEND_ERROR[rf]	/	12 ¹	/	/	12	12	12 ¹
SEND_EXPEDITED_DATA	/	-	-	-	-	-	-
SET_SYNCPT_OPTIONS[ok]	-	-	-	-	-	-	-
SYNCPT[ok]	/	-	5	1	/	/	/
SYNCPT[bo]	/	*	*	*	/	/	/
TEST(P)[ok]	/	/	/	/	-	/	/
TEST(P)[ae]	/	/	/	/	12	/	/
TEST(P)[bo]	/	/	/	/	*	/	/
TEST(P)[da]	/	/	/	/	12	/	/
TEST(P)[dn]	/	/	/	/	12	/	/
TEST(P)[en]	/	/	/	/	-	/	/
TEST(P)[ep]	/	/	/	/	-	/	/
TEST(P)[et]	/	/	/	/	-	/	/
TEST(P)[me]	/	/	/	/	2	/	/
TEST(P)[pn]	/	/	/	/	-	/	/
TEST(P)[rf]	/	/	/	/	12	/	/
TEST(P)[un]	/	/	/	/	-	/	/
TEST(Q)[ok]	/	-	-	-	-	/	/
TEST(Q)[un]	/	-	-	-	-	/	/
WAIT[ok]	/	/	/	/	-	/	/
WAIT[ae]	/	/	/	/	12	/	/
WAIT[bo]	/	/	/	/	*	/	/
WAIT[da]	/	/	/	/	12 ¹	/	/
WAIT[dn]	/	/	/	/	12	/	/
WAIT[en]	/	/	/	/	-	/	/
WAIT[ep]	/	/	/	/	-	/	/
WAIT[et]	/	/	/	/	-	/	/
WAIT[pn]	/	/	/	/	-	/	/
WAIT[me]	/	/	/	/	2	/	/
WAIT[rf]	/	/	/	/	12 ¹	/	/

Figure E-3. Half-duplex Conversation State Transition Matrix (Part 3 of 3)

Conversation State Matrices

Conversation States (continued)							Verb
Confirm Dealloc- cate 8	Sync- point 9	Sync- point Send 10	Sync- point Dealloc- cate 11	Dealloc- cate 12	Backout Requir- ed 13	Dealloc- cate Pending 14	
/	/	/	/	/	/	/	SEND_DATA[ok]
/	/	/	/	/	/	/	SEND_DATA[ae]
/	/	/	/	/	/	/	SEND_DATA[bo]
/	/	/	/	/	/	/	SEND_DATA[da]
/	/	/	/	/	/	/	SEND_DATA[ep]
/	/	/	/	/	/	/	SEND_DATA[me]
/	/	/	/	/	/	/	SEND_DATA[rf]
2	2	2	2	/	/	/	SEND_ERROR[ok] ²
/	/	/	/	/	/	/	SEND_ERROR[ok] ³
/	/	/	/	/	/	/	SEND_ERROR[ae]
/	/	/	/	/	/	/	SEND_ERROR[bo]
/	/	/	/	/	/	/	SEND_ERROR[da]
/	/	/	/	/	/	/	SEND_ERROR[dn]
/	/	/	/	/	/	/	SEND_ERROR[ep]
/	/	/	/	/	/	/	SEND_ERROR[me]
12 ¹	12 ¹	12 ¹	12 ¹	/	/	/	SEND_ERROR[rf]
-	-	-	-	/	-	/	SEND_EXPEDITED_DATA
-	-	-	-	-	-	/	SET_SYNCPT_OPTIONS
/	5	2	12	/	/	/	SYNCPT[ok]
/	*	*	*	/	*	/	SYNCPT[bo]
/	/	/	/	/	/	/	TEST(P)[ok]
/	/	/	/	/	/	/	TEST(P)[ae]
/	/	/	/	/	/	/	TEST(P)[bo]
/	/	/	/	/	/	/	TEST(P)[da]
/	/	/	/	/	/	/	TEST(P)[dn]
/	/	/	/	/	/	/	TEST(P)[en]
/	/	/	/	/	/	/	TEST(P)[ep]
/	/	/	/	/	/	/	TEST(P)[et]
/	/	/	/	/	/	/	TEST(P)[me]
/	/	/	/	/	/	/	TEST(P)[pn]
/	/	/	/	/	/	/	TEST(P)[rf]
/	/	/	/	/	/	/	TEST(P)[un]
/	/	/	/	/	/	/	TEST(Q)[ok]
/	/	/	/	/	/	/	TEST(Q)[un]
/	/	/	/	/	/	/	WAIT[ok]
/	/	/	/	/	/	/	WAIT[ae]
/	/	/	/	/	/	/	WAIT[bo]
/	/	/	/	/	/	/	WAIT[da]
/	/	/	/	/	/	/	WAIT[dn]
/	/	/	/	/	/	/	WAIT[en]
/	/	/	/	/	/	/	WAIT[ep]
/	/	/	/	/	/	/	WAIT[et]
/	/	/	/	/	/	/	WAIT[me]
/	/	/	/	/	/	/	WAIT[pn]
/	/	/	/	/	/	/	WAIT[rf]

¹ Other protected conversations of the local program may be in the backout required state following this verb. Check (MC_)GET_ATTRIBUTES CONVERSATION_STATE to find out if this is so.

² SEND_ERROR[ok]² shows what is allowed when the verb follows a RECEIVE verb with a WHAT_RECEIVED value of TAKE_SYNCPT*.

³ SEND_ERROR[ok]³ shows what is allowed when the verb follows a PREPARE_FOR_SYNCPT verb.

The only differences between cases ² and ³ occur in the sync point states.

Conversation State Matrices

Parameter Abbreviations (...) A TYPE(ABEND_PROG), TYPE(ABEND_SVC), or TYPE(ABEND_TIMER) C TYPE(CONFIRM), or TYPE(SYNC_LEVEL) with synchronization level CONFIRM F TYPE(FLUSH) L TYPE(LOCAL) P TEST(POSTED) Q TEST(REQUEST TO SEND RECEIVED) S TYPE(SYNC_LEVEL) with synchronization level SYNCPT	Return-Code Abbreviations [...] ae ALLOCATION_ERROR bo BACKED_OUT da DEALLOCATE_ABEND_PROG, DEALLOCATE_ABEND_SVC, or DEALLOCATE_ABEND_TIMER dn DEALLOCATE_NORMAL en PROG_ERROR_NO_TRUNC or SVC_ERROR_NO_TRUNC ep PROG_ERROR_PURGING or SVC_ERROR_PURGING et PROG_ERROR_TRUNC or SVC_ERROR_TRUNC me MAP_EXECUTION_ERROR or MAP_NOT_FOUND or MAPPING_NOT_SUPPORTED or USER_CONTROL_DATA_NOT_SUPPORTED oi OPERATION_INCOMPLETE ok OK pe PARAMETER_ERROR pn POSTING_NOT_ACTIVE rf RESOURCE_FAILURE_NO_RETRY or RESOURCE_FAILURE_RETRY un UNSUCCESSFUL
What-Received Abbreviations {...} co CONFIRM cd CONFIRM_DEALLOCATE cs CONFIRM_SEND da DATA, DATA_COMPLETE, DATA_INCOMPLETE, or LL_TRUNCATED sd TAKE_SYNCPT_DEALLOCATE se SEND ss TAKE_SYNCPT_SEND sy TAKE_SYNCPT	Matrix Symbols / Verb cannot be issued in this state - Remain in current state number Number of next state * State at completion of most recent synchronization point

Figure E-4. Legend for Half-duplex Conversation State Transition Matrices

Conversation State Matrices

Verb	Conversation States						
	Reset	Send	Defer Receive	Defer Deallo- cate	Receive	Confirm	Confirm Send
	1	2	3	4	5	6	7
Initiating Conversation	/	/	/	/	/	/	/
ALLOCATE	/	/	/	/	/	/	/
BACKOUT	/	/	/	/	/	/	/
CONFIRM	/	X	/	►	►	►	►
CONFIRMED	/	►	►	►	►	/	/
DEALLOCATE(F)	/	X	►	►	►	►	►
DEALLOCATE(C)	/	X	►	►	►	►	►
DEALLOCATE(S)	/	X	►	►	►	►	►
DEALLOCATE(A)	/	/	/	/	/	/	/
DEALLOCATE(L)	/	►	►	►	►	►	►
FLUSH	/	/	/	►	►	►	►
GET_ATTRIBUTES	/	/	/	/	/	/	/
GET_TP_PROPERTIES	/	/	/	/	/	/	/
GET_TYPE	/	/	/	/	/	/	/
POST_ON_RECEIPT	/	►	►	►	/	►	►
PREPARE_FOR_SYNCPT	/	/	/	/	►	►	►
PREPARE_TO_RECEIVE(F)	/	X	►	►	►	►	►
PREPARE_TO_RECEIVE(C)	/	X	/	/	►	►	►
PREPARE_TO_RECEIVE(S)	/	X	►	►	►	►	►
RECEIVE_AND_WAIT	/	X	►	►	/	►	►
RECEIVE_EXPEDITED_DATA	/	/	/	/	/	/	/
RECEIVE_IMMEDIATE	/	►	►	►	/	►	►
REQUEST_TO_SEND	/	/	►	►	/	/	/
SEND_DATA	/	/	►	►	►	►	►
SEND_ERROR	/	/	►	►	/	/	/
SEND_EXPEDITED_DATA	/	/	/	/	/	/	/
SET_SYNCPT_OPTIONS	/	/	/	/	/	/	/
SYNCPT	/	X	/	/	►	►	►
TEST(P)	/	►	►	►	/	►	►
TEST(Q)	/	/	/	/	/	►	►
WAIT	/	/	/	/	/	/	/
Parameter Abbreviations (...) A TYPE(ABEND_PROG), TYPE(ABEND_SVC), or TYPE(ABEND_TIMER) C TYPE(CONFIRM), or TYPE(SYNC_LEVEL) with synchronization level CONFIRM F TYPE(FLUSH) L TYPE(LOCAL) P TEST(POSTED) Q TEST(REQUEST_TO_SEND_RECEIVED) S TYPE(SYNC_LEVEL) with synchronization level SYNCPT							

Figure E-5. Half-duplex Conversation State Check Matrix

Conversation State Matrices

Conversation States (continued)							Verb
Confirm Deallo- cate 8	Sync- point 9	Sync- point Send 10	Sync- point Deallo- cate 11	Deallo- cate 12	Backout Requi- red 13	Deallo- cate Pending 14	
/	/	/	/	/	/	/	Initiating Conversation
/	/	/	/	/	▶	/	ALLOCATE
/	/	/	/	/	/	/	BACKOUT
▶	▶	▶	▶	▶	▶	/	CONFIRM
/	▶	▶	▶	▶	▶	/	CONFIRMED
▶	▶	▶	▶	▶	▶	/	DEALLOCATE(F)
▶	▶	▶	▶	▶	▶	/	DEALLOCATE(C)
▶	▶	▶	▶	▶	▶	/	DEALLOCATE(S)
/	/	/	/	▶	▶	/	DEALLOCATE(A)
▶	▶	▶	▶	/	▶	/	DEALLOCATE(L)
▶	▶	▶	▶	▶	▶	/	FLUSH
/	/	/	/	/	/	/	GET_ATTRIBUTES
/	/	/	/	/	/	/	GET_TP_PROPERTIES
/	/	/	/	/	/	/	GET_TYPE
▶	▶	▶	▶	▶	▶	/	POST_ON_RECEIPT
▶	▶	▶	▶	▶	▶	/	PREPARE_FOR_SYNCPT
▶	▶	▶	▶	▶	▶	/	PREPARE_TO_RECEIVE(F)
▶	▶	▶	▶	▶	▶	/	PREPARE_TO_RECEIVE(C)
▶	▶	▶	▶	▶	▶	/	PREPARE_TO_RECEIVE(S)
▶	▶	▶	▶	▶	▶	/	RECEIVE_AND_WAIT
/	/	/	/	/	/	▶	RECEIVE_EXPEDITED_DATA
▶	▶	▶	▶	▶	▶	/	RECEIVE_IMMEDIATE
/	/	/	▶	▶	▶	▶	REQUEST_TO_SEND
▶	▶	▶	▶	▶	▶	/	SEND_DATA
/	/	/	/	▶	▶	/	SEND_ERROR
/	/	/	/	/	/	▶	SEND_EXPEDITED_DATA
/	/	/	/	/	/	/	SET_SYNCPT_OPTIONS
▶	/	/	/	▶	/	/	SYNCPT
▶	▶	▶	▶	▶	▶	/	TEST(P)
▶	▶	▶	▶	▶	▶	/	TEST(Q)
/	/	/	/	/	▶	/	WAIT

Matrix Symbols

- ▶ State check ABEND condition occurs in this state
- X State check ABEND condition occurs in this state if the program is in the process of sending a logical record and the record is incomplete
- / State check ABEND condition cannot occur in this state

Full-Duplex Conversation State Matrices

The following full-duplex conversation states are defined at the conversation protocol boundary (where the prefix [MC_] in a verb name means the verb can be either a mapped or basic conversation verb):

Reset — the state in which the program can allocate the conversation.

Send-receive — the state in which the program can send data *and* receive information from the remote program.

Send-only — the state in which the program can send data; but cannot receive information from the remote program, because the local receive queue has been closed by receipt of a DEALLOCATE_NORMAL return code.

Receive-only — the state in which the program can receive information from the remote program; but cannot send data, because the local program has issued a (MC_)DEALLOCATE verb to close its send queue.

Parameter Abbreviations (...) A TYPE(ABEND_PROG), TYPE(ABEND_SVC), or TYPE(ABEND_TIMER) D RESOURCE_ID(DISCARD) F TYPE(FLUSH)	Return-Code Abbreviations [...] ae ALLOCATION_ERROR da DEALLOCATE_ABEND_PROG, DEALLOCATE_ABEND_SVC, or DEALLOCATE_ABEND_TIMER dn DEALLOCATE_NORMAL ei ERROR_INDICATION en PROG_ERROR_NO_TRUNC or SVC_ERROR_NO_TRUNC et PROG_ERROR_TRUNC or SVC_ERROR_TRUNC me MAP_EXECUTION_ERROR or MAP_NOT_FOUND or MAPPING_NOT_SUPPORTED or USER_CONTROL_DATA_NOT_SUPPORTED (mapped conversations only) ok OK pe PARAMETER_ERROR rf RESOURCE_FAILURE_NO_RETRY or RESOURCE_FAILURE_RETRY un UNSUCCESSFUL
What-Received Abbreviations {...} da DATA, DATA_COMPLETE, DATA_INCOMPLETE, or LL_TRUNCATED	Matrix Symbols / Verb cannot be issued in this state - Remain in current state n Number of next state

Figure E-6. Legend for Full-Duplex Conversation State Transition Matrix

Conversation State Matrices

Verb	Conversation States			
	Reset	Send-Receive	Send-Only	Receive-Only
	1	2	3 *	4
Initiating Conversation	2	/	/	/
ALLOCATE[ok]	2	/	/	/
ALLOCATE[ae]	-	/	/	/
ALLOCATE[pe]	-	/	/	/
ALLOCATE[un]	-	/	/	/
DEALLOCATE(D,F)[ok]	/	4	1	/
DEALLOCATE(D,F)[ei]	/	4	1	/
DEALLOCATE(D,A)[ok]	/	1	1	1
Flush[ok]	/	-	-	/
Flush[ei]	/	-	-	/
GET_ATTRIBUTES	/	-	-	-
GET_TP_PROPERTIES	-	-	-	-
GET_TYPE	/	-	-	-
RECEIVE_AND_WAIT[ok]{da}	/	-	/	-
RECEIVE_AND_WAIT[ae]	/	1	/	1
RECEIVE_AND_WAIT[da]	/	1	/	1
RECEIVE_AND_WAIT[dn]	/	3	/	1
RECEIVE_AND_WAIT[en]	/	-	/	-
RECEIVE_AND_WAIT[et]	/	-	/	-
RECEIVE_AND_WAIT[me]	/	-	/	-
RECEIVE_AND_WAIT[rf]	/	1	/	1
RECEIVE_EXPEDITED_DATA	/	-	-	-
RECEIVE_IMMEDIATE[ok]{da}	/	-	/	-
RECEIVE_IMMEDIATE[ae]	/	1	/	1
RECEIVE_IMMEDIATE[da]	/	1	/	1
RECEIVE_IMMEDIATE[dn]	/	3	/	1
RECEIVE_IMMEDIATE[en]	/	-	/	-
RECEIVE_IMMEDIATE[et]	/	-	/	-
RECEIVE_IMMEDIATE[me]	/	-	/	-
RECEIVE_IMMEDIATE[rf]	/	1	/	1
RECEIVE_IMMEDIATE[un]	/	-	/	-
SEND_DATA[ok]	/	-	-	/
SEND_DATA[ei]	/	-	1	/
SEND_DATA[me]	/	-	-	/
SEND_ERROR[ok]	/	-	-	/
SEND_ERROR[ei]	/	-	1	/
SEND_EXPEDITED_DATA	/	-	-	-

Figure E-7. Full-Duplex Conversation State Transition Matrix

Verb	Conversation States			
	Reset	Send-Receive	Send-Only	Receive-Only
	1	2	3	4
Initiating Conversation	/	/	/	/
ALLOCATE	/	/	/	/
DEALLOCATE(F)	/	/	X	►
DEALLOCATE(A)	/	/	/	/
FLUSH	/	/	/	►
GET_ATTRIBUTES	/	/	/	/
GET_TP_PROPERTIES	/	/	/	/
GET_TYPE	/	/	/	/
RECEIVE_AND_WAIT	/	/	►	/
RECEIVE_EXPEDITED_DATA	/	/	/	/
RECEIVE_IMMEDIATE	/	/	►	/
SEND_DATA	/	/	/	►
SEND_ERROR	/	/	/	►
SEND_EXPEDITED_DATA	/	/	/	/
<u>Parameter Abbreviations (...)</u> A TYPE(ABEND_PROG), TYPE(ABEND_SVC), or TYPE(ABEND_TIMER) F TYPE(FLUSH), or TYPE(SYNC_LEVEL) with synchronization level NONE <u>Matrix Symbols</u> ► State check occurs in this state X State check occurs in this state if the program is in the process of sending a logical record and the record is incomplete / State check cannot occur in this state				

Figure E-8. Full-Duplex Conversation State Check Matrix

Nonblocking State Matrices

When the transaction program issues a nonblocking verb, two situations can occur. When the required resources for processing the verb are available, the verb completes and produces all the return values. The queues incur no state changes. If any of the required resources is not available, the verb is suspended and placed in the associated queue. The queue, then, enters the *outstanding* state.

Depending on the queueing discipline, the outstanding state of a queue can influence the processing of the verbs issued to it. When the queueing discipline is FIFO, the verbs belonging to the queue are processed and completed in the same order as issued by the transaction program. Therefore, an incoming request from the transaction program does not jump ahead of the outstanding verbs already in the queue. For example, an incoming RECEIVE_AND_WAIT verb will not receive information that is intended for an outstanding RECEIVE_AND_WAIT verb currently in the queue. In this case, the outstanding state of the queue is used to govern the processing order of the verbs issued by the local program, as opposed to guarding against whether or not the verb can be issued in a conversation state. On the other hand, the outstanding state has no significance to the FIRO queues, since the verbs on those queues can complete in any order.

Since multiple verbs may be outstanding at the same time for a conversation, the transaction program can still issue verbs to a conversation when it is in the middle of deallocation. To prevent the conversation verbs from hanging after the conversation has been deallocated, the conversation queues enter the *prohibited* state when the conversation is being deallocated. The prohibited state, indicating the conversation will soon be gone, is used to clean up the conversation queues and turn down any incoming requests from the transaction program for that conversation.

The following states are defined for the conversation queues:

Empty — the state in which the conversation queue does not have any outstanding verb operations.

Outstanding — the state in which the conversation queue has at least one suspended verb operation. Verbs issued to the queue in this state will receive a return code of OPERATION_INCOMPLETE.

Prohibited — the state in which the conversation queue can no longer handle any incoming request from the transaction program. Verbs issued to the queue in this state will receive a return code of PROGRAM_PARAMETER_CHECK.

A state transition for a conversation queue can occur as a result of an issuance or a completion of a verb from the local program. A conversation queue enters outstanding state when the needed resources are not available for processing a verb. All the conversation queues enter prohibited state at the same time when the local program issues the DEALLOCATE verb of type FLUSH, ABEND, and CONFIRM with a received CONFIRMED indication in the half-duplex conversation, and of type ABEND and FLUSH in the send-only state in the full-duplex conversation. Regardless of blocking or nonblocking, when a verb completes, the associated queue can change to the *empty* state if the verb is the only one in the queue.

Conversation State Matrices

The state transitions in each of the conversation queues are represented by means of a matrix, see Figure E-9 on page E-17. Since the queues do not perform a state check on the verbs, the state-check matrix does not exist.

When a verb is issued by the local program, the LU first examines the state matrices for the queues. Then, the verb will be subject to the state checking with respect to the half-duplex or full-duplex conversation state matrix when the verb is being executed. All parameters are validated at verb execution time; the resource ID is also validated at verb issuing time. Additional parameter type checking may be done at verb issuing time for those systems that have the ability to do so.

Verb	Queue states		
	Empty 1	Outstanding 2	Prohibited 3
nonblocking verb with RC set to OPERATION_INCOMPLETE	2	-	/
nonblocking verb with RC set to OPERATION_NOT_ACCEPTED	/	-	/
nonblocking verb set to any completion indication and last verb in its queue	-	1	/
DEALLOCATE ¹	3	3	/
Note: The notational conventions are the same as described for the conversation state matrices.			

¹ Transitions to prohibited state can be caused by issuance of the DEALLOCATE verb with TYPE=FLUSH, TYPE=ABEND, and TYPE=CONFIRM (when CONFIRMED is received) in a half-duplex conversation, and with TYPE=ABEND and TYPE=FLUSH (when in send-only state) in a full-duplex conversation.

Figure E-9. Queue State-Transition Matrix

Conversation State Matrices

Appendix F. Conversation Return Codes

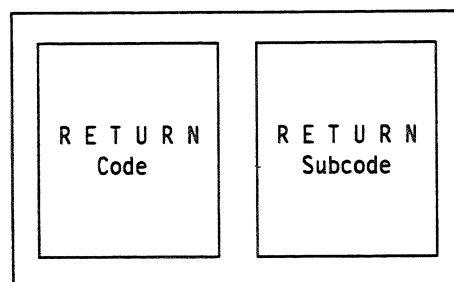
Purpose of Conversation Return Code

Some verbs have a parameter called `RETURN_CODE` used to pass a return code back to the program at the completion of the LU's execution of a verb. The return code indicates the result of verb execution, including any state changes to the specified conversation. See Appendix E, "Conversation State Matrices" on page E-1 for a definition of the states.

Some of the return codes indicate results of the local LU's processing of a verb; these return codes are returned on the verb that invoked the local processing. Other return codes indicate results of processing invoked at the remote end of the conversation and, depending on the verb, can be returned on the verb that invoked the remote processing or on a subsequent verb. Still other return codes report events that originate at the remote end of the conversation. In any case, only one code is returned at a time. Other verb-specific information may be passed back in verb-unique parameters. See each specific verb for a description of any verb-unique parameters.

Some of the return codes have the suffix "`RETRY`" or "`NO_RETRY`" in their name. `RETRY` means that the condition indicated by the return code may not be permanent, and the program may attempt to allocate the conversation again. Whether the retry attempt succeeds depends on the duration of the condition. In general, the program should limit the number of times it attempts to retry without success, after which it should consider the condition permanent. `NO_RETRY` means that the condition is most likely permanent, and, in general, no attempt should be made to allocate the conversation again until the condition is corrected.

List of Conversation Return Codes and Subcodes



The return codes are described below. Each description includes the meaning of the return code, the origin of the condition indicated by the return code, when the return code can be reported to the program, and the state of the conversation when control is returned to the program. The individual verb descriptions list the applicable return codes.

Conversation Return Codes

Return code descriptions for both half-duplex and full-duplex conversations:

ALL_UNKNOWN_WAIT_OBJECTS indicates the local program issued a **WAIT_FOR_COMPLETION** verb and none of the wait objects in the specified **WAIT_OBJECT_LIST** was identifiable. The states of the conversation and associated queues remain unchanged.

ALLOCATION_ERROR indicates the local program issued an **MC_ALLOCATE** or **ALLOCATE** verb and allocation of the specified conversation could not be completed. The **ALLOCATION_ERROR** indication together with one of the following subcodes form the complete return code that is returned to the program; the subcode identifies the specific error. (The remote LU and remote program referred to in the following subcode definitions are the LU named on the **LU_NAME** parameter and the program named on the **TPN** parameter, respectively, of the verb.)

ALLOCATION_ERROR can have the following subcodes:

- **ALLOCATION_FAILURE_NO_RETRY** indicates the conversation cannot be allocated on a session because of a condition that is not temporary. For example, the session to be used for the conversation cannot be activated because the current (LU,mode) session limit for the specified (LU-name,mode-name) pair is 0, or because of a system definition error or a session-activation protocol error; or the session was deactivated because of a session protocol error before the conversation could be allocated. The program should not retry the allocation request until the condition is corrected. This return code is returned on the **MC_ALLOCATE** or **ALLOCATE** verb when the program specifies (by means of the **RETURN_CONTROL** parameter) that the local LU is to attempt to allocate a session before returning control to the program; otherwise, it is returned on a subsequent verb.
- **ALLOCATION_FAILURE_RETRY** indicates the conversation cannot be allocated on a session because of a condition that may be temporary. For example, the session to be used for the conversation cannot be activated because of a temporary lack of resources at the local LU or remote LU; or the session was deactivated because of session outage before the conversation could be allocated. The condition may be temporary, and the program can retry the allocation request. This return code is returned on the **MC_ALLOCATE** or **ALLOCATE** verb when the program specifies (by means of the **RETURN_CONTROL** parameter) that the local LU is to attempt to allocate a session before returning control to the program; otherwise, it is returned on a subsequent verb.
- **PIP_NOT_ALLOWED** indicates the remote LU rejected the allocation request because the local program specified program initialization parameters (by means of the **PIP(YES)** parameter) and either the remote LU does not support PIP data, or the remote program has no PIP variables defined (see &INT62. for a discussion of transaction program structure and initialization data). This return code is returned on a subsequent verb.
- **PIP_NOT_SPECIFIED_CORRECTLY** indicates the remote LU rejected the allocation request because the remote program has one or more PIP

variables defined and the local program specified no program initialization parameters (by means of the PIP(NO) parameter), or it specified program initialization parameters (by means of the PIP(YES) parameter) that do not correspond in number to those defined for the remote program. This return code is returned on a subsequent verb.

- **SECURITY_NOT_VALID** indicates the remote LU rejected the allocation request because the access security information (specified by means of the SECURITY parameter) is invalid. This return code is returned on a subsequent verb.
- **SYNC_LEVEL_NOT_SUPPORTED_BY_LU** indicates the local LU rejected the allocation request because the local program specified a synchronization level (by means of the SYNC_LEVEL parameter) that the remote LU does not support. This return code is returned on the MC_ALLOCATE or ALLOCATE verb when the program specifies (by means of the RETURN_CONTROL parameter) that the local LU is to attempt to allocate a session before returning control to the program; otherwise, it is returned on a subsequent verb.
- **SYNC_LEVEL_NOT_SUPPORTED_BY_PGM** indicates the remote LU rejected the allocation request because the local program specified a synchronization level (by means of the SYNC_LEVEL parameter) that the remote program does not support. This return code is returned on a subsequent verb.
- **TPN_NOT_RECOGNIZED** indicates the remote LU rejected the allocation request because the local program specified a remote program name that the remote LU does not recognize. This return code is returned on a subsequent verb.
- **TP_NOT_AVAIL_NO_RETRY** indicates the remote LU rejected the allocation request because the local program specified a remote program that the remote LU recognizes but cannot start. The condition is not temporary, and the program should not retry the allocation request. This return code is returned on a subsequent verb.
- **TP_NOT_AVAIL_RETRY** indicates the remote LU rejected the allocation request because the local program specified a remote program that the remote LU recognizes but currently cannot start. The condition may be temporary, and the program can retry the allocation request. This return code is returned on a subsequent verb.

With one exception, the subcodes of ALLOCATION_ERROR are not explicitly listed in the individual verb descriptions, as any of them can be returned as part of the ALLOCATION_ERROR return code. The exception to this is for the MC_ALLOCATE and ALLOCATE verbs, on which only certain subcodes can be returned; therefore, the applicable subcodes are listed for these verbs.

CONVERSATION_ENDED indicates the local LU received notification from the remote LU that the conversation is ending, or the local LU itself is ending the conversation. The conversation may be ending normally or terminating because of an error condition.

CONVERSATION_ENDED is returned to the expedited-send and expedited-receive queues only. The return code indicating the cause of conversation ter-

Conversation Return Codes

mination is returned to the receive queue. No state change occurs because of receipt of the **CONVERSATION_ENDED** return code; however, the expedited queues are effectively closed and future requests will be rejected with **CONVERSATION_ENDED** until the conversation state transitions to reset.

ENCRYPTION_NOT_SUPPORTED indicates the local program issued an **MC_SEND_DATA** or **SEND_DATA** specifying the parameter **ENCRYPT(YES)**, but the mode does not support either mandatory or selective data encryption, so the send fails. The state of the conversation remains unchanged.

EXPEDITED_DATA_NOT_SUPPORTED_BY_LU indicates the local LU rejected the send or receive expedited data request because the local program issued an expedited data verb that the remote LU does not support.

INSUFFICIENT_BUFFER_SIZE_PROVIDED indicates the local program issued a verb containing a buffer size that is insufficient for the amount of expedited data that the local program is to receive.

INVALID_TIMEOUT_VALUE indicates the local program issued a **WAIT_FOR_COMPLETION** verb and the **TIMEOUT** parameter contained a value that exceeds the range permitted by the implementation. The states of the conversation and associated queues remain unchanged.

OK indicates the verb issued by the local program executed successfully. That is, the function defined for the verb, up to the point at which control is returned to the program, was performed as specified. The state of the conversation is as defined for the verb.

For some verbs, the **OK** indication together with one of the following subcodes form the complete return code that is returned to the program; the subcode provides additional information.

- **DATA** indicates data is available for the program to receive.

OPERATION_INCOMPLETE indicates that the nonblocking verb issued by the local program has been processed by the LU but has not yet completed. This return code can be reported on the verb specifying the null or nonnull wait object. The state of the conversation remains unchanged; the state of the associated queue is in *outstanding* state.

OPERATION_NOT_ACCEPTED indicates that the nonblocking verb issued by the local program was not processed by the LU because the number of verbs that can be enqueued has reached the implementation limit. The condition may be temporary, and the program can retry the request. This return code is reported on the verb specifying the null or nonnull wait object when the associated queue cannot enqueue any more requests. The states of the conversation and associated queues remain unchanged.

PARAMETER_ERROR indicates the local program issued a verb specifying a parameter containing an invalid argument. The source of the argument is considered to be outside the program definition, such as an LU name supplied by a terminal operator and used as the argument of **LU_NAME** on **MC_ALLOCATE** or **ALLOCATE**. Contrast this definition with the definition of the Parameter Check return code in the section "Programming Error

Conditions" on page 4-5. This return code is returned on the verb specifying the invalid argument. The state of the conversation remains unchanged.

PROGRAM_PARAMETER_CHECK indicates the local program issued a verb containing an invalid value for a parameter. This return code can be reported to the local program if the conversation specified by a verb is currently being deallocated as the result of an outstanding DEALLOCATE operation.

PROGRAM_STATE_CHECK indicates the local program issued a verb that is not valid in the current program state.

TIMER_EXPIRED indicates the local program issued a **WAIT_FOR_COMPLETION** verb and none of the wait objects specified in the **WAIT_OBJECT_LIST** has been posted when the timer expired. The states of the conversation and associated queues remain unchanged.

UNSUCCESSFUL indicates the verb issued by the local program did not execute successfully. This return code is returned on the unsuccessful verb. The state of the conversation remains unchanged.

USER_CONTROL_DATA_NOT_SUPPORTED indicates the local program issued an **MC_SEND_DATA** specifying that the data record contains user control data (by means of the **USER_CONTROL_DATA** parameter), and either the remote LU or remote program does not support user control data. This return code is reported on a subsequent verb. All information sent by the local program on the **MC_SEND_DATA** verb and subsequent verbs prior to the reporting of the **USER_CONTROL_DATA_NOT_SUPPORTED** return code is purged. The state of the conversation remains unchanged.

Return code descriptions for half-duplex conversations only:

ALLOCATION_ERROR indicates the local program issued an MC_ALLOCATE or ALLOCATE verb and allocation of the specified conversation could not be completed. The ALLOCATION_ERROR indication together with one of the following subcodes form the complete return code that is returned to the program; the subcode identifies the specific error. (The remote LU and remote program referred to in the following subcode definitions are the LU named on the LU_NAME parameter and the program named on the TPN parameter, respectively, of the verb.) When ALLOCATION_ERROR (with any subcode) is returned to the program, the conversation is in deallocate state.

ALLOCATION_ERROR can have the following subcodes:

- **CONVERSATION_TYPE_MISMATCH** indicates the remote LU rejected the allocation request because the local program issued an MC_ALLOCATE or ALLOCATE verb and the remote program does not support the respective mapped- or basic-conversation protocol boundary, or the local program issued an MC_ALLOCATE verb and the remote LU does not support mapped conversations. This return code is returned on a subsequent verb.

BACKED_OUT indicates the remote program issued a BACKOUT, or the local or remote LU has done so, in order to restore all protected resources to their status as of the last synchronization point. This return code can be reported to the local program on a verb it issues in send, defer, receive, or syncpoint state. It is also returned when protected resources are in backout-required state.

This return code is not returned to the BACKOUT verb; The OK return code to the BACKOUT verb indicates that the protected resources have been backed out.

When the return code is received on any verb but SYNCPT, the conversation is in backout-required state. When it is received on the SYNCPT verb, the conversation is in the state that it was in immediately following the last synchronization point, i.e., the state following the completion of the last successfully sync point operation if one has occurred, otherwise the state at the start of the conversation.

On most verbs, the BACKED_OUT return code has no subcode. On the SYNCPT and (MC_)PREPARE_FOR_SYNCPT verbs, BACKED_OUT can have the following subcodes, which show what happened to the rest of the distributed transaction affected by the verb. For the SYNCPT verb, this means all of the protected resources in the distributed transaction. For the (MC_)PREPARE_FOR_SYNCPT verb, this means the protected resources subordinate to the conversation partner, i.e., to which it propagates the sync point operation.

- **ALL_AGREED** indicates that all protected resources have backed out in the part of the distributed transaction affected by the verb.

- **LUW_OUTCOME_PENDING** indicates that as a result of failures it is not yet known whether all protected resources have backed out in the part of the distributed transaction affected by the verb. At least one program in the part of the distributed transaction must have specified **SET_SYNCPT_OPTIONS WAIT_FOR_OUTCOME(NO)** in order for the program to get this subcode.
- **LUW_OUTCOME_MIXED** indicates that at least one resource in the part of the distributed transaction affected by the verb committed as a result of operator intervention. The distributed transaction is damaged.

DEALLOCATE_ABEND indicates the remote program issued an **MC_DEALLOCATE** verb specifying the **TYPE(ABEND)** parameter, or the remote LU has done so because of a remote program ABEND condition. If the conversation for the remote program was in receive state when the verb was issued, information sent by the local program and not yet received by the remote program is purged. This return code can be reported to the local program on a verb it issues in send, defer, or receive state. The conversation is in deallocate state.

For conversations where the **SYNC_LEVEL** is **SYNCPT**, receipt of this primary return code may or may not put the TP in a state where backout is required, depending on when the underlying event occurred relative to the last **SYNCPT** or **BACKOUT** operation. A secondary return code is provided to indicate to the TP whether or not backout is required.

- **BACKOUT_REQUIRED** indicates that backout is required. A TP receiving this subcode enters the backout-required state.
- **BACKOUT_NOT_REQUIRED** indicates that backout is not required.

DEALLOCATE_ABEND_PROG indicates the remote program issued a **DEALLOCATE** verb specifying the **TYPE(ABEND_PROG)** parameter, or the remote LU has done so because of a remote program ABEND condition. If the conversation for the remote program was in receive state when the verb was issued, information sent by the local program and not yet received by the remote program is purged. This return code can be reported to the local program on a verb it issues in send, defer, or receive state. The conversation is in deallocate state.

For conversations where the **SYNC_LEVEL** is **SYNCPT**, receipt of this primary return code may or may not put the TP in a state where backout is required, depending on when the underlying event occurred relative to the last **SYNCPT** or **BACKOUT** operation. A secondary return code is provided to indicate to the TP whether or not backout is required.

- **BACKOUT_REQUIRED** indicates that backout is required. A TP receiving this subcode enters the backout-required state.
- **BACKOUT_NOT_REQUIRED** indicates that backout is not required.

This return code is typically used by transaction programs while the **DEALLOCATE_ABEND_SVC** return code is typically used by the LU. This allows for distinguishing between transaction program errors and system errors.

Conversation Return Codes

DEALLOCATE_ABEND_SVC indicates the remote program issued a DEALLOCATE verb specifying the TYPE(ABEND_SVC) parameter. If the conversation for the remote program was in receive state when the verb was issued, information sent by the local program and not yet received by the remote program is purged. This return code can be reported to the local program on a verb it issues in send, defer, or receive state. The conversation is in deallocate state.

For conversations where the SYNC_LEVEL is SYNCPT, receipt of this primary return code may or may not put the TP in a state where backout is required, depending on when the underlying event occurred relative to the last SYNCPT or BACKOUT operation. A secondary return code is provided to indicate to the TP whether or not backout is required.

- **BACKOUT_REQUIRED** indicates that backout is required. A TP receiving this subcode enters the backout-required state.
- **BACKOUT_NOT_REQUIRED** indicates that backout is not required.

This return code is typically used by the LU while the **DEALLOCATE_ABEND_PROG** return code is typically used by transaction programs. This allows for distinguishing between system errors and transaction program errors.

DEALLOCATE_ABEND_TIMER indicates the remote program issued a DEALLOCATE verb specifying the TYPE(ABEND_TIMER) parameter. If the conversation for the remote program was in receive state when the verb was issued, information sent by the local program and not yet received by the remote program is purged. This return code can be reported to the local program on a verb it issues in send, defer, or receive state. The conversation is in deallocate state.

For conversations where the SYNC_LEVEL is SYNCPT, receipt of this primary return code may or may not put the TP in a state where backout is required, depending on when the underlying event occurred relative to the last SYNCPT or BACKOUT operation. A secondary return code is provided to indicate to the TP whether or not backout is required.

- **BACKOUT_REQUIRED** indicates that backout is required. A TP receiving this subcode enters the backout-required state.
- **BACKOUT_NOT_REQUIRED** indicates that backout is not required.

DEALLOCATE_NORMAL indicates the remote program issued an MC_DEALLOCATE or DEALLOCATE verb specifying the TYPE(SYNC_LEVEL) or TYPE(FLUSH) parameter; if TYPE(SYNC_LEVEL), either the synchronization level is NONE or it is SYNCPT and the remote program subsequently issued an MC_FLUSH or FLUSH verb. This return code is reported to the local program on a verb it issues in receive state. The conversation is in deallocate state.

For conversations where the SYNC_LEVEL is SYNCPT, receipt of this primary return code may or may not put the TP in a state where backout is required, depending on when the underlying event occurred relative to the last SYNCPT or BACKOUT operation. A secondary return code is provided to indicate to the TP whether or not backout is required.

- **BACKOUT_REQUIRED** indicates that backout is required. A TP receiving this subcode enters the backout-required state.
- **BACKOUT_NOT_REQUIRED** indicates that backout is not required.

MAP_EXECUTION_FAILURE indicates the local program issued an **MC_SEND_DATA** specifying the data record is to be mapped (by means of the **MAP_NAME** parameter), and the local LU or the remote LU could not map the data record based on the map name. This return code is returned on the **MC_SEND_DATA** verb when the map execution failed at the local LU. Otherwise, the remote LU rejects the data, and the return code is reported on a subsequent verb. All information sent by the local program on that **MC_SEND_DATA** verb and subsequent verbs prior to the reporting of the **MAP_EXECUTION_FAILURE** return code is purged. The conversation is in send state.

MAP_NOT_FOUND indicates the local program issued an **MC_SEND_DATA** specifying the data record is to be mapped (by means of the **MAP_NAME** parameter), and the map name is unknown to the local LU or the remote LU. This return code is reported on the **MC_SEND_DATA** verb when the map name is unknown to the local LU. Otherwise, the remote LU rejects the data, and the return code is reported on a subsequent verb. All information sent by the local program on that **MC_SEND_DATA** verb and subsequent verbs prior to the reporting of the **MAP_NOT_FOUND** return code is purged. The conversation is in send state.

MAPPING_NOT_SUPPORTED indicates the local program issued an **MC_SEND_DATA** specifying the data record is to be mapped (by means of the **MAP_NAME** parameter), and either the remote LU or remote program does not support data mapping. This return code is reported on a subsequent verb. All information sent by the local program on that **MC_SEND_DATA** verb and subsequent verbs prior to the reporting of the **MAPPING_NOT_SUPPORTED** return code is purged. The conversation is in send state.

OK indicates the verb issued by the local program executed successfully. That is, the function defined for the verb, up to the point at which control is returned to the program, was performed as specified. The state of the conversation is as defined for the verb.

For some verbs, the **OK** indication together with one of the following subcodes form the complete return code that is returned to the program; the subcode provides additional information.

- **DATA** indicates data is available for the program to receive.
- **NOT_DATA** indicates information other than data is available for the program to receive.

On the **SYNCPT**, **BACKOUT**, and **(MC_)PREPARE_FOR_SYNCPT** verbs, **OK** can have the following subcodes, which show what happened to the rest of the distributed transaction affected by the verb. For the **BACKOUT** and **SYNCPT** verbs, this means all of the protected resources in the distributed transaction. For the **(MC_)PREPARE_FOR_SYNCPT** verb, this means the protected resources subordinate to the conversation partner, i.e., to which it propagates the sync point operation.

Conversation Return Codes

- **ALL_AGREED** indicates that all protected resources in the part of the distributed transaction affected by the verb have performed the action requested by the verb. For the SYNCPT verb, they have all committed. For the (MC_)PREPARE_FOR_SYNCPT verb, they have all prepared to commit. For the BACKOUT verb, they have all backed out.
- **VOTED_READ_ONLY** indicates that all protected resources in the part of the distributed transaction affected by the verb have voted read only, including the local resources. In order for a program to receive this return code, it must have previously issued the SET_SYNCPT_OPTIONS verb with VOTE_READ_ONLY_PERMITTED(YES). This subcode is only returned for the SYNCPT verb.
- **LUW_OUTCOME_PENDING** indicates that as a result of failures it is not yet known whether all protected resources in the part of the distributed transaction affected by the verb have performed the same action. At least one program in the part of the distributed transaction must have specified SET_SYNCPT_OPTIONS WAIT_FOR_OUTCOME(NO) in order for the program to get this subcode.
- **LUW_OUTCOME_MIXED** indicates that at least one resource in the part of the distributed transaction affected by the verb has backed out and one other has committed as a result of operator intervention. The distributed transaction is damaged.

POSTING_NOT_ACTIVE indicates the local program issued a verb that determines whether a resource has been posted, and posting is not active for any of the specified resources.

PROG_ERROR_NO_TRUNC indicates one of the following:

- The remote program issued an MC_SEND_ERROR verb and the conversation for the remote program was in send state. No truncation occurs at the mapped conversation protocol boundary. This return code is reported to the local program on an MC_RECEIVE_AND_WAIT or MC_RECEIVE_IMMEDIATE verb it issues prior to receiving any data records or after receiving one or more data records. It can also be received on an MC_PREPARE_FOR_SYNCPT verb, indicating that the remote program issued the MC_SEND_ERROR verb in response to a TAKE_SYNCPT_* indication on a MC_RECEIVE_AND_WAIT or MC_RECEIVE_IMMEDIATE verb.
- The remote program issued a SEND_ERROR verb specifying the TYPE(PROG) parameter, the conversation for the remote program was in send state, and the verb did not truncate a logical record. No truncation occurs at the basic conversation protocol boundary when a program issues SEND_ERROR before sending any logical records or after sending a complete logical record. This return code is reported to the local program on a RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb it issues prior to receiving any logical records or after receiving one or more complete logical records. It can also be received on a PREPARE_FOR_SYNCPT verb, indicating that the remote program issued the SEND_ERROR verb in response to a TAKE_SYNCPT_* indication on a RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb.

The conversation remains in receive state.

PROG_ERROR_PURGING indicates the remote program issued an MC_SEND_ERROR verb or it issued a SEND_ERROR verb specifying the TYPE(PROG) parameter, and the conversation for the remote program was in receive, confirm, or sync point state. The verb may have caused information to be purged. Purging occurs when a program issues MC_SEND_ERROR or SEND_ERROR in receive state before receiving all the information sent by its partner program, that is, all the information sent prior to the reporting of the PROG_ERROR_PURGING return code to the partner program. The purging can occur at the local LU, remote LU, or both. No purging occurs when the remote program issues the (MC_)SEND_ERROR verb in confirm state or sync point state, or in receive state after receiving all the information sent by its partner program. This return code is normally reported to the local program on a verb it issues after sending some information to the remote program. However, the return code can be reported on a verb the program issues prior to sending any information, depending on the verb and when it is issued. The conversation is in receive state.

PROG_ERROR_PURGING can also be received on a (MC_)PREPARE_FOR_SYNCPT verb, indicating that the remote program issued the SEND_ERROR verb before receiving the TAKE_SYNCPT_* indication on a RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb.

PROG_ERROR_TRUNC indicates the remote program issued a SEND_ERROR verb specifying the TYPE(PROG) parameter, the conversation for the remote program was in send state, and the verb truncated a logical record. Truncation occurs at the basic conversation protocol boundary when a program begins sending a logical record and then issues SEND_ERROR before sending the complete logical record. This return code is reported to the local program on a RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb it issues after receiving the truncated logical record. The conversation remains in receive state.

RESOURCE_FAILURE_NO_RETRY indicates a failure occurred that caused the conversation to be prematurely terminated. For example, the session being used for the conversation was deactivated because of a session protocol error, or the conversation was deallocated because of a protocol error between the mapped conversation components of the LUs. The condition is not temporary, and the program should not retry the transaction until the condition is corrected. This return code can be reported to the local program on a verb it issues in any state other than reset or deallocate. The conversation is in deallocate state.

For conversations where the SYNC_LEVEL is SYNCPT, receipt of this primary return code may or may not put the TP in a state where backout is required, depending on when the underlying event occurred relative to the last SYNCPT or BACKOUT operation. A secondary return code is provided to indicate to the TP whether or not backout is required.

- **BACKOUT_REQUIRED** indicates that backout is required. A TP receiving this subcode enters the backout-required state.
- **BACKOUT_NOT_REQUIRED** indicates that backout is not required.

Conversation Return Codes

RESOURCE_FAILURE_RETRY indicates a failure occurred that caused the conversation to be prematurely terminated. For example, the session being used for the conversation was deactivated because of a session outage, such as a line failure, a modem failure, or a crypto engine failure. The condition may be temporary, and the program can retry the transaction. This return code can be reported to the local program on a verb it issues in any state other than reset or deallocate. The conversation is in deallocate state.

For conversations where the **SYNC_LEVEL** is **SYNCPT**, receipt of this primary return code may or may not put the TP in a state where backout is required, depending on when the underlying event occurred relative to the last **SYNCPT** or **BACKOUT** operation. A secondary return code is provided to indicate to the TP whether or not backout is required.

- **BACKOUT_REQUIRED** indicates that backout is required. A TP receiving this subcode enters the backout-required state.
- **BACKOUT_NOT_REQUIRED** indicates that backout is not required.

SVC_ERROR_NO_TRUNC indicates the remote program issued a **SEND_ERROR** verb specifying the **TYPE(SVC)** parameter. Otherwise, this return code, as it applies to the basic conversation protocol boundary, has the same meaning as **PROG_ERROR_NO_TRUNC**. The conversation remains in receive state.

SVC_ERROR_PURGING indicates the remote program issued a **SEND_ERROR** verb specifying the **TYPE(SVC)** parameter. Otherwise, this return code, as it applies to the basic conversation protocol boundary, has the same meaning as **PROG_ERROR_PURGING**. The conversation is in receive state.

SVC_ERROR_TRUNC indicates the remote program issued a **SEND_ERROR** specifying the **TYPE(SVC)** parameter. Otherwise, this return code has the same meaning as **PROG_ERROR_TRUNC**. The conversation remains in receive state.

Return code descriptions for full-duplex conversations only:

ALLOCATION_ERROR indicates the local program issued an MC_ALLOCATE or ALLOCATE verb and allocation of the specified conversation could not be completed. The ALLOCATION_ERROR indication together with one of the following subcodes form the complete return code that is returned to the program; the subcode identifies the specific error. (The remote LU and remote program referred to in the following subcode definitions are the LU named on the LU_NAME parameter and the program named on the TPN parameter, respectively, of the verb.) When ALLOCATION_ERROR (with any subcode) is returned to the program's receive queue, the conversation is in reset state.

ALLOCATION_ERROR can have the following subcodes:

- **CONVERSATION_TYPE_MISMATCH** indicates the remote LU rejected the allocation request because the local program issued an MC_ALLOCATE or ALLOCATE verb and the remote program does not support the conversation type specified (by means of the TYPE parameter). This return code is returned on a subsequent verb.
- **FULL_DUPLEX_NOT_SUPPORTED_BY_LU** indicates the local LU rejected the allocation request because the remote program does not support full-duplex conversations. This return code is returned on the MC_ALLOCATE or ALLOCATE verb when the program specifies (by means of the RETURN_CONTROL parameter) that the local LU is to attempt to allocate a session before returning control to the program; otherwise, it is returned on a subsequent verb.

DEALLOCATE_ABEND indicates the remote program issued an MC_DEALLOCATE verb specifying the TYPE(ABEND) parameter, or the remote LU has done so because of a remote program ABEND condition. If the conversation for the remote program was in receive-only state when the verb was issued, information sent by the local program and not yet received by the remote program is purged. This return code can be reported to the local program's receive queue. The conversation is in reset state.

DEALLOCATE_ABEND_PROG indicates the remote program issued a DEALLOCATE verb specifying the TYPE(ABEND_PROG) parameter, or the remote LU has done so because of a remote program ABEND condition. If the conversation for the remote program was in receive-only state when the verb was issued, information sent by the local program and not yet received by the remote program is purged. This return code can be reported to the local program's receive queue. The conversation is in deallocate state.

This return code is typically used by transaction programs while the DEALLOCATE_ABEND_SVC return code is typically used by the LU. This allows for distinguishing between transaction program errors and system errors.

DEALLOCATE_ABEND_SVC indicates the remote program issued a DEALLOCATE verb specifying the TYPE(ABEND_SVC) parameter. If the conversation for the remote program was in receive-only state when the verb was issued, information sent by the local program and not yet received by the

Conversation Return Codes

remote program is purged. This return code can be reported to the local program's receive queue. The conversation is in reset state.

This return code is typically used by the LU, while the **DEALLOCATE_ABEND_PROG** return code is typically used by transaction programs. This allows for distinguishing between system errors and transaction program errors.

DEALLOCATE_ABEND_TIMER indicates the remote program issued a **DEALLOCATE** verb specifying the **TYPE(ABEND_TIMER)** parameter. If the conversation for the remote program was in receive-only state when the verb was issued, information sent by the local program and not yet received by the remote program is purged. This return code can be reported to the local program's receive queue. The conversation is in reset state.

DEALLOCATE_NORMAL indicates the remote program issued an **MC_DEALLOCATE** or **DEALLOCATE** verb specifying the **TYPE(SYNC_LEVEL)** or **TYPE(FLUSH)** parameter to effectively close its send queue. This return code is reported to the local program's receive queue. The conversation is in send-only state and may continue to issue send queue verbs until the local program is ready to close its send queue and thereby end the conversation. In a full-duplex conversation each program must issue an **(MC_)DEALLOCATE** to close its send queue; therefore, two **(MC_)DEALLOCATE** verbs (one by the local program and one by the remote program) are required to end a full-duplex conversation.

ERROR_INDICATION indicates the local LU received a terminating error notification from the remote LU, and the requested send queue operation has not been performed. A terminating error is an error that causes the conversation to end. It may be an **ALLOCATION_ERROR**, **DEALLOCATE_ABEND_***, or a conversation failure. The nature of the terminating error is reflected in the **ERROR_INDICATION** subcode.

The **ERROR_INDICATION** return code is returned to send queue verbs only, and gives an early notification to the send queue to stop sending data because the conversation is ending. If the current conversation state is send-receive, receipt of **ERROR_INDICATION** does not cause a state change; however, the send queue is effectively closed because future send queue verbs will be rejected with the **ERROR_INDICATION** return code. The conversation will not actually end until the receive queue receives the terminating error or the local program issues an **(MC_)DEALLOCATE_ABEND_***.

If the current conversation state is send-only, receipt of the **ERROR_INDICATION** return code will change the conversation state to reset and end the conversation.

ERROR_INDICATION may have the following subcodes:

- **ALLOCATION_ERROR** indicates the remote LU rejected the allocation request. Additional information regarding the nature of the **ALLOCATION_ERROR** will be reported to the local program's receive queue, in the form of an **ALLOCATION_ERROR** subcode.
- **DEALLOCATE_ABEND** indicates the remote program issued an **MC_DEALLOCATE** verb specifying the **TYPE(ABEND)** parameter, or the remote LU has done so because of a remote program **ABEND** condi-

tion. The **DEALLOCATE_ABEND** return code will be returned to the local program's receive queue.

- **DEALLOCATE_ABEND_PROG** indicates the remote program issued an **DEALLOCATE** verb specifying the **TYPE(ABEND_PROG)** parameter, or the remote LU has done so because of a remote program **ABEND** condition. The **DEALLOCATE_ABEND_PROG** return code will be returned to the local program's receive queue.
- **DEALLOCATE_ABEND_SVC** indicates the remote program issued an **DEALLOCATE** verb specifying the **TYPE(ABEND_SVC)** parameter. The **DEALLOCATE_ABEND_SVC** return code will be returned to the local program's receive queue.
- **DEALLOCATE_ABEND_TIMER** indicates the remote program issued an **DEALLOCATE** verb specifying the **TYPE(ABEND_TIMER)** parameter. The **DEALLOCATE_ABEND_TIMER** return code will be returned to the local program's receive queue.
- **RESOURCE_FAILURE_NO_RETRY** indicates a failure occurred at the remote LU that caused the conversation to be prematurely terminated. It may be that the session being used for the conversation was deactivated because of a session protocol error, or the conversation was deallocated because of a protocol layer between the components of the LUs. The condition is not temporary, and the program should not retry the transaction until the condition is corrected. The **RESOURCE_FAILURE_NO_RETRY** return code will be returned to the local program's receive queue.
- **RESOURCE_FAILURE_RETRY** indicates a failure occurred at the remote LU that caused the conversation to be prematurely terminated. It may be that the session being used for the conversation was deactivated because of a link failure or a modem failure. The condition may be temporary, and the program can retry the transaction. The **RESOURCE_FAILURE_RETRY** return code will be returned to the local program's receive queue.
- **UNKNOWN** indicates the local LU does not recognize the type of terminating error received from the remote LU.

MAP_EXECUTION_FAILURE indicates the local program issued an **MC_SEND_DATA** specifying the data record is to be mapped (by means of the **MAP_NAME** parameter), and the local LU or the remote LU could not map the data record based on the map name. This return code is returned on the **MC_SEND_DATA** verb when the map execution failed at the local LU. Otherwise, the remote LU rejects the data, and the return code is reported on a subsequent verb. All information sent by the local program on that **MC_SEND_DATA** verb and subsequent verbs prior to the reporting of the **MAP_EXECUTION_FAILURE** return code is purged. The state of the conversation remains unchanged.

MAP_NOT_FOUND indicates the local program issued an **MC_SEND_DATA** specifying the data record is to be mapped (by means of the **MAP_NAME** parameter), and the map name is unknown to the local LU or the remote LU. This return code is reported on the **MC_SEND_DATA** verb when the map name is unknown to the local LU. Otherwise, the remote LU rejects the data,

Conversation Return Codes

and the return code is reported on a subsequent verb. All information sent by the local program on that MC_SEND_DATA verb and subsequent verbs prior to the reporting of the MAP_NOT_FOUND return code is purged. The state of the conversation remains unchanged.

MAPPING_NOT_SUPPORTED indicates the local program issued an MC_SEND_DATA specifying the data record is to be mapped (by means of the MAP_NAME parameter), and either the remote LU or remote program does not support data mapping. This return code is reported on a subsequent verb. All information sent by the local program on that MC_SEND_DATA verb and subsequent verbs prior to the reporting of the MAPPING_NOT_SUPPORTED return code is purged. The state of the conversation remains unchanged.

PROG_ERROR_NO_TRUNC indicates one of the following:

- The remote program issued an MC_SEND_ERROR verb. No truncation occurs at the mapped conversation protocol boundary. This return code is reported to the local program on a receive queue verb it issues prior to receiving any data records or after receiving one or more data records.
- The remote program issued a SEND_ERROR verb specifying the TYPE(PROG) parameter and the verb did not truncate a logical record. No truncation occurs at the basic conversation protocol boundary when a program issues SEND_ERROR before sending any logical records or after sending a complete logical record. This return code is reported to the local program on a receive queue verb it issues prior to receiving any logical records or after receiving one or more complete logical records.

The state of the conversation remains unchanged.

PROG_ERROR_TRUNC indicates the remote program issued a SEND_ERROR verb specifying the TYPE(PROG) parameter and the verb truncated a logical record. Truncation occurs at the basic conversation protocol boundary when a program begins sending a logical record and then issues SEND_ERROR before sending the complete logical record. This return code is reported to the local program on a receive queue verb it issues after receiving the truncated logical record. The state of the conversation remains unchanged.

RESOURCE_FAILURE_NO_RETRY indicates a failure occurred that caused the conversation to be prematurely terminated. For example, the session being used for the conversation was deactivated because of a session protocol error, or the conversation was deallocated because of a protocol error between the mapped conversation components of the LUs. The condition is not temporary, and the program should not retry the transaction until the condition is corrected. This return code can be reported to the local program's receive queue. The conversation is in reset state.

RESOURCE_FAILURE_RETRY indicates a failure occurred that caused the conversation to be prematurely terminated. For example, the session being used for the conversation was deactivated because of a session outage, such as a line failure, a modem failure, or a cryptography device failure. The condition may be temporary, and the program can retry the transaction. This return code can be reported to the local program's receive queue. The conversation is in reset state.

SVC_ERROR_NO_TRUNC indicates the remote program issued a **SEND_ERROR** verb specifying the **TYPE(SVC)** parameter. Otherwise, this return code, as it applies to the basic conversation protocol boundary, has the same meaning as **PROG_ERROR_NO_TRUNC**. The state of the conversation remains unchanged.

SVC_ERROR_TRUNC indicates the remote program issued a **SEND_ERROR** specifying the **TYPE(SVC)** parameter. Otherwise, this return code has the same meaning as **PROG_ERROR_TRUNC**. The state of the conversation remains unchanged.

Correlation of Conversation Verbs and Return Codes

Figure F-1 on page F-18 shows the correlation of the return codes to the verbs on which they can be returned. The "X" in the figure means the return code can be returned on the corresponding verb. A verb without any "X"s beside it means no return codes are defined for the verb.

Conversation Return Codes

	Return Codes												
	ALL UNKNOWNT WARRANTS OBJECTS	ALL LOCATIONS ERRORS OBJECTS	CONVERTED ERRORS OBJECTS	BACKLOG ERRORS OBJECTS	DEALLOCATE ABEND PROG	DEALLOCATE ABEND SVC	DEALLOCATE ABEND TIMER	DEALLOCATE ABEND TIMER	DEALLOCATE ABEND TIMER	ENCRYPTION NOT SUPPORTED	EXPIRED DATA NOT SUPPORTED	INSUFFICIENT BUFFERS PROVIDED	INVALID FORMAT
Verbs													
MC_ALLOCATE		X		X	X							X	X
MC_CONFIRM		X		X	X							X	X
MC_CONFIRMED												X	X
MC_DEALLOCATE		X		X								X	X
MC_FLUSH												X	X
MC_GET_ATTRIBUTES												X	X
MC_POST_ON_RECEIPT												X	X
MC_PREPARE_FOR_SYNCPT		X		X	X							X	X
MC_PREPARE_TO_RECEIVE		X		X	X							X	X
MC_RECEIVE_AND_WAIT		X		X	X			X				X	X
MC_RECEIVE_EXPEDITED_DATA		X	X	X	X					X	X	X	X
MC_RECEIVE_IMMEDIATE		X		X	X			X				X	X
MC_REQUEST_TO_SEND		X	X	X	X							X	X
MC_SEND_DATA		X		X	X				X			X	X
MC_SEND_ERROR		X		X	X			X				X	X
MC_SEND_EXPEDITED_DATA		X		X	X					X		X	X
MC_TEST		X		X	X			X				X	X
BACKOUT													X
GET_TP_PROPERTIES													X
GET_TYPE													X
SYNCPT				X	X	X	X	X	X			X	X
WAIT		X		X	X	X	X	X	X			X	X
WAIT_FOR_COMPLETION	X											X	X
ALLOCATE		X											X
CONFIRM		X		X									X
CONFIRMED													X
DEALLOCATE		X											X
FLUSH													X
GET_ATTRIBUTES													X
POST_ON_RECEIPT													X
PREPARE_FOR_SYNCPT		X		X									X
PREPARE_TO_RECEIVE		X		X									X
RECEIVE_AND_WAIT		X		X									X
RECEIVE_EXPEDITED_DATA		X	X	X						X	X		X
RECEIVE_IMMEDIATE		X		X									X
REQUEST_TO_SEND		X		X									X
SEND_DATA		X		X					X				X
SEND_ERROR		X		X									X
SEND_EXPEDITED_DATA		X		X						X			X
TEST		X		X									X

Figure F-1. Correlation of Half-Duplex Conversation Return Codes to Verbs (1 of 2)

Conversation Return Codes

[illegible]

Figure F-2. Correlation of Half-Duplex Conversation Return Codes to Verbs (2 of 2)

Conversation Return Codes

	Return Codes																
	ALL UNKNOWN WAIT OBJECTS	ALL LOCATION ERROR	CONVERSION ENDED	DEALLOCATE ABEND	DEALLOCATE ABEND PROG	DEALLOCATE ABEND SVC	DEALLOCATE ABEND TIMER	DEALLOCATE NORMAL	ENCRYPTION NOT SUPPORTED	ERROR INDICATION	EXPEDITED DATA NOT SUPPORTED BY LU	INSUFFICIENT BUFFER SIZE PROVIDED	INVALID TIMEOUT VALUE	MAP EXECUTION FAILURE	MAP NOT FOUND	MAPPING NOT SUPPORTED	OK
Verbs																	
MC_ALLOCATE MC_DEALLOCATE MC_FLUSH MC_GET_ATTRIBUTES MC_RECEIVE_AND_WAIT MC_RECEIVE_EXPEDITED_DATA MC_RECEIVE_IMMEDIATE MC_SEND_DATA MC_SEND_ERROR MC_SEND_EXPEDITED_DATA		X X X X	 X X	 X X	 												

Figure F-3. Correlation of Full-Duplex Conversation Return Codes to Verbs (1 of 2)

Conversation Return Codes

Return Codes													
OPERATION INCOMPLETE	OPERATION NOT ACCEPTED	PARAMETER ERROR	PROGRAM ERROR NOT TRUNC	PROGRAM ERROR TRUNC	PROGRAM PARAMETER CHECK	PROGRAM STATE CHECK	RESOURCE FAILURE NO RETRY	RESOURCE FAILURE RETRY	SVC ERROR NOT TRUNC	SVC ERROR TRUNC	TIMER EXPIRED	USER CONTROL DATA NOT SUPPORTED	UNSUCCESSFUL
													Verbs
XXXX	XXXX	X			XXXXXX	XX						X	MC_ALLOCATE MC_DEALLOCATE MC_FLUSH MC_GET_ATTRIBUTES MC_RECEIVE_AND_WAIT MC_RECEIVE_EXPEDITED_DATA MC_RECEIVE_IMMEDIATE MC_SEND_DATA MC_SEND_ERROR MC_SEND_EXPEDITED_DATA
XXXXXX	XXXXXX		X		XXXXXX	X	X	X				X	X
					XX						X		GET_TP_PROPERTIES GET_TYPE WAIT_FOR_COMPLETION
XXXX	XXXX	X			XXXXXX	XX						X	ALLOCATE DEALLOCATE FLUSH GET_ATTRIBUTES RECEIVE_AND_WAIT RECEIVE_EXPEDITED_DATA RECEIVE_IMMEDIATE SEND_DATA SEND_ERROR SEND_EXPEDITED_DATA
XXXXXX	XXXXXX		X	X	XXXXXX	X	X	X	X	X		X	X

Figure F-4. Correlation of Full-Duplex Conversation Return Codes to Verbs (2 of 2)

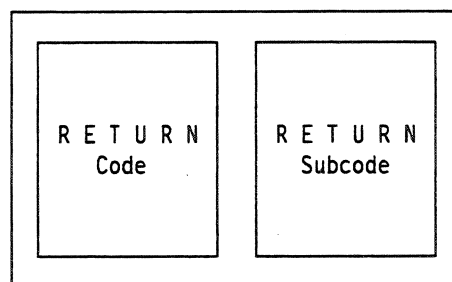
Conversation Return Codes

Appendix G. Control-Operator Return Codes

Purpose of Control Operator Return Code

Some verbs have a parameter called `RETURN_CODE` used to pass a return code back to the transaction program at the completion of the LU's execution of a verb. The return code indicates the result of processing the verb on which it is returned. Only one code is returned at a time. Other verb-specific information may be passed back in verb-unique parameters. See each specific verb for a description of any verb-unique parameters.

List of Control Operator Return Codes and Subcodes



The return codes are described below. Each description includes the meaning of the return code and the origin of the condition indicated by the return code.

ACTIVATION_FAILURE_NO_RETRY indicates the `ACTIVATE_SESSION` verb failed to activate the session because of a condition that is not temporary. For example, the session cannot be activated because the (LU,mode) session limit for the specified target LU and mode name is currently 0 at the target LU—this applies to single sessions and to sessions for the `SNASVCMG` mode name; or because of a system definition error or a session-activation protocol error. The control operator should not retry the transaction until the condition is corrected.

ACTIVATION_FAILURE_RETRY indicates the `ACTIVATE_SESSION` verb failed to activate the session because of a temporary condition. For example, the session cannot be activated because of a temporary lack of resources at the source LU or target LU. The control operator may retry the session activation later.

ALLOCATION_ERROR indicates the `CNOS` verb did not execute successfully because the allocation of the control operator conversation with the target LU cannot be completed. The `ALLOCATION_ERROR` indication together with one of the following subcodes form the complete return code that is returned to the transaction program; the subcode identifies the specific error. The source and target LUs' `CNOS` parameters are not changed.

Control-Operator Return Codes

- **ALLOCATION_FAILURE_NO_RETRY** indicates the control operator conversation cannot be allocated because of a condition that is not temporary. For example, the session to be used for the control operator conversation cannot be activated because the (LU,mode) session limit for the specified target LU and SNASVCMG mode name is currently 0 at either the source LU or target LU; or because of a system definition error or a session-activation protocol error; or because a session protocol error caused the session to be deactivated before the conversation could be allocated. The control operator should not retry the transaction until the condition is corrected.
- **ALLOCATION_FAILURE_RETRY** indicates the control operator conversation cannot be allocated because of a temporary condition. For example, the session to be used for the control operator conversation cannot be activated because of a temporary lack of resources at the source LU or target LU; or the session was deactivated because of session outage before the conversation could be allocated. The condition is temporary, and the control operator can retry the transaction later.
- **TP_NOT_AVAIL_RETRY** indicates the target LU is currently unable to start the transaction program identified as hex 06F1, which is the SNA service transaction program for the control operator. For example, there may be a temporary lack of resources the target LU needs to start the transaction program. The condition is temporary, and the control operator can retry the transaction later.

COMMAND_RACE_REJECT indicates the CNOS verb did not execute successfully because the source LU or target LU is currently processing another CNOS transaction for the same mode name. The other CNOS transaction is processed to completion. The source and target LUs' CNOS parameters are not changed by the unsuccessful CNOS verb.

LU_MODE_SESSION_LIMIT_CLOSED indicates the CNOS verb did not execute successfully because the target LU currently will not allow the (LU,mode) session limit for the specified mode name to be raised above 0. The (LU,mode) session limit remains at 0. This condition is not necessarily permanent; the control operator may retry the CNOS transaction later.

LU_MODE_SESSION_LIMIT_EXCEEDED indicates the **ACTIVATE_SESSION** verb could not activate the session with the specified mode name to the target LU, for one of the following reasons:

1. For a single session connection to the target LU, either the (LU,mode) session limit is currently 0, or an LU-LU session is already active (with the specified or a different mode name).
2. For a parallel session connection to the target LU, the number of currently active sessions with the specified mode name equals the (LU,mode) session limit.

LU_MODE_SESSION_LIMIT_NOT_ZERO indicates the program attempted to initialize an (LU,mode) session limit that is already initialized, that is, the session limit is already greater than 0. The source and target LUs' CNOS parameters are not changed.

LU_MODE_SESSION_LIMIT_ZERO indicates the program attempted to change an (LU,mode) session limit that has not been initialized, that is, the session limit is 0. The source and target LUs' CNOS parameters are not changed.

LU_SESSION_LIMIT_EXCEEDED indicates the CNOS verb did not execute successfully because the new (LU,mode) session limit would cause the sum of the (LU,mode) session limits to exceed the total LU-LU session limit for the source LU (see the **DEFINE_LOCAL_LU** verb). The sum of the (LU,mode) session limits is calculated as follows:

1. A single session connection to a target LU is counted as 1 if at least one of the (LU,mode) session limits for that target LU is 1, including the specified session limit. Otherwise, it is counted as 0.
2. A parallel session connection to a target LU is counted as the sum of all (LU,mode) session limits for the target LU, including the specified session limit.

OK indicates the verb executed successfully. The following subcodes augment this return code and indicate whether the parameter values were processed as specified or as negotiated by the target LU:

- **AS_SPECIFIED** indicates the two LUs executed the verb as specified, without negotiation.
- **AS_NEGOTIATED** indicates the two LUs executed the verb as negotiated by the target LU. One or more parameter values have been negotiated. The transaction program can obtain the negotiated parameter values by issuing the **DISPLAY_MODE** verb. The verb descriptions define which parameter values can be negotiated. A special case of negotiation exists for intra-LU sessions. See note 6 on page 8-6.
- **FORCED** indicates the source LU forced the resetting of its (LU,mode) session limit as a result of an error condition that prevented successful completion of the exchange of the CNOS request and reply. The target LU's CNOS parameters may not be changed, depending on the error condition and when it occurred during the CNOS exchange.

PARAMETER_ERROR indicates the verb did not execute successfully because it specifies a parameter that contains an invalid argument. The source of the argument may be outside the transaction program definition, such as a control-operator supplied LU name or mode name. When this return code is returned on a CNOS verb, the source and target LUs' CNOS parameters are not changed. When it is returned on a session activation or deactivation verb, the LU-LU session is not activated or deactivated, respectively. When it is returned on a define, display, or delete verb, the LU operating parameters are not altered or returned.

REQUEST_EXCEEDS_MAX_ALLOWED indicates the CNOS verb did not execute successfully because it specifies an (LU,mode) session limit that exceeds the source LU's maximum (LU,mode) session limit defined for the target LU and mode name (see the **DEFINE_MODE** verb). The source and target LUs' CNOS parameters are not changed.

Control-Operator Return Codes

RESOURCE_FAILURE_NO_RETRY indicates on the CNOS verb that it did not execute successfully because of a failure that caused the control operator conversation to be prematurely deallocated. For example, the session being used for the control operator conversation was deactivated because of a session protocol error, or because of session outage from which the control operator component of the LU could not recover; or the conversation was deallocated because of a protocol error between the control operator components of the LUs. The condition is not temporary, and the control operator should not retry the transaction until the condition is corrected. The CNOS parameters remain unchanged at the source LU, or both the source and target LUs, depending on when the failure occurred.

On the SIGNOFF verb, which may involve allocation requests to partner LUs, this return code may have the same meaning as specified for conversation verbs.

RESOURCE_FAILURE_RETRY on the SIGNOFF verb indicates conditions similar to those described for conversation verbs.

UNRECOGNIZED_MODE_NAME indicates the CNOS verb did not execute successfully because the target LU does not recognize the specified mode name. The source and target LUs' CNOS parameters are not changed.

Correlation of Control Operator Verbs and Return Codes

Figure G-1 on page G-6 shows the correlation of the return codes to the verbs on which they can be returned. The "X" in the figure means the return code can be returned on the corresponding verb. A verb without any "X"s under it means no return codes are defined for the verb. The individual verb descriptions list the applicable return codes. However, the subcodes of **ALLOCATION_ERROR** are not explicitly listed, as any of them can be returned as part of the **ALLOCATION_ERROR** return code.

Control-Operator Return Codes

Control-Operator Return Codes

	Return Codes						
	A C T I V A T I O N F A I L U R E N O R E T R Y	A C T I V A T I O N F A I L U R E R E T R Y	A L L O C A T I O N E R R O R	C O M M A N D R A C E R E J E C T	L U M O D E S E S S I O N L I M I T C L O S E D	L U M O D E S E S S I O N L I M I T E X C E E D E D	L U M O D E S E S S I O N L I M I T N O T Z E R O
Verbs							
CHANGE_SESSION_LIMIT			X	X			
INITIALIZE_SESSION_LIMIT			X	X	X		X
PROCESS_SESSION_LIMIT							
RESET_SESSION_LIMIT			X	X	X		
ACTIVATE_SESSION	X	X				X	
DEACTIVATE_CONVERSATION_GROUP							
DEACTIVATE_SESSION							
DEFINE_LOCAL_LU							
DEFINE_REMOTE_LU							
DEFINE_MODE							
DEFINE_TP							
DISPLAY_LOCAL_LU							
DISPLAY_REMOTE_LU							
DISPLAY_MODE							
DISPLAY_TP							
DELETE							
DISPLAY_SIGNED_ON_LIST							
PROCESS_SIGN_OFF							
SIGNOFF							

Figure G-1. Correlation of Control Operator Return Codes to Verbs

Control-Operator Return Codes

Return Codes								Verbs
LU MODE SESSION LIMIT ZERO	LU SESSION LIMIT EXCEEDED	OK	PARAMETER ERROR	REQUEST EXCEEDS MAX ALLOWED	RESOURCE FAILURE NO RETRY	RESOURCE FAILURE RETRY	UNRECOGNIZED MODE NAME	
X	X X	X X X X	X X X	X X	X X X		X	CHANGE_SESSION_LIMIT INITIALIZE_SESSION_LIMIT PROCESS_SESSION_LIMIT RESET_SESSION_LIMIT
		X X X	X X					ACTIVATE_SESSION DEACTIVATE_CONVERSATION_GROUP DEACTIVATE_SESSION
		X X X X X X X X X X	X X X X X X X X					DEFINE_LOCAL_LU DEFINE_REMOTE_LU DEFINE_MODE DEFINE_TP DISPLAY_LOCAL_LU DISPLAY_REMOTE_LU DISPLAY_MODE DISPLAY_TP DELETE
		X X X	X		X X	X		DISPLAY_SIGNED_ON_LIST PROCESS_SIGN_OFF SIGN_OFF

Figure G-2. Correlation of Control Operator Return Codes to Verbs

Control-Operator Return Codes

Index

A

access security

- already verified indication 5-5, 7-5
- for conversation 7-5
- for mapped conversation 5-5
- NONE 5-5
- PGM 5-5
- SAME 5-5

access, resource 10-22

accumulating data

- in LU's receive buffer 1-10
- in LU's send buffer 1-9

ACTION_IF_PROBLEMS parameter

- of SET_SYNCPT_OPTIONS verb 6-10

ACTIVATE_SESSION verb 9-2

ACTIVATION_FAILURE_NO_RETRY return code

- for control operator verbs G-1

ACTIVATION_FAILURE_RETRY return code

- for control operator verbs G-1

ADDITIONAL_CRITERIA parameter

- of SIGNOFF verb 11-8

Advanced Program-to-Program Communication (APPC) 1-3

ALL_AGREED

- See BACKED_OUT subcodes
- See OK subcodes

ALL_UNKNOWN_WAIT_OBJECTS return code

ALLOCATE verb 7-2

- used by LU services component 7-9
- used by transaction program 7-9

allocation

- of a conversation 7-2
- of a mapped conversation 5-2
- of an LU-LU session 5-2, 7-2

ALLOCATION_ERROR

- See ERROR_INDICATION subcodes

ALLOCATION_ERROR return code

- for control operator verbs G-1
- for conversation verbs F-2, F-6, F-13

ALLOCATION_ERROR subcodes

- for control operator verbs
 - ALLOCATION_FAILURE_NO_RETRY G-2
 - ALLOCATION_FAILURE_RETRY G-2
 - TP_NOT_AVAIL_RETRY G-2

- for conversation verbs

- ALLOCATION_FAILURE_NO_RETRY F-2
- ALLOCATION_FAILURE_RETRY F-2
- CONVERSATION_TYPE_MISMATCH F-6, F-13
- FULL_DUPLEX_NOT_SUPPORTED_BY_LU F-13
- PIP_NOT_ALLOWED F-2

ALLOCATION_ERROR subcodes (continued)

for conversation verbs (continued)

- PIP_NOT_SPECIFIED_CORRECTLY F-2
- SECURITY_NOT_VALID F-3
- SYNC_LEVEL_NOT_SUPPORTED_BY_LU F-3
- SYNC_LEVEL_NOT_SUPPORTED_BY_PGM F-3
- TP_NOT_AVAIL_NO_RETRY F-3
- TP_NOT_AVAIL_RETRY F-3
- TPN_NOT_RECOGNIZED F-3

ALLOCATION_FAILURE_NO_RETRY

- See ALLOCATION_ERROR subcodes

ALLOCATION_FAILURE_RETRY

- See ALLOCATION_ERROR subcodes

allocation request

- carrying transaction program name 1-8

allocation requests, draining 8-17

already verified 10-17

already verified indication 5-5, 7-5

APPC

- See Advanced Program-to-Program Communication (APPC)

application transaction program 5-1, 6-1

arguments, parameter

- See parameter, arguments

AS_NEGOTIATED

- See OK subcodes

AS_SPECIFIED

- See OK subcodes

authorization list

- resource access 10-22

AV_AND_PV 10-17

B

BACKED_OUT return code F-6

BACKED_OUT subcodes

for sync-point verbs

- ALL_AGREED 5-35, 6-13, 7-37, F-6
- LUW_OUTCOME_MIXED 5-35, 6-13, 7-37, F-7
- LUW_OUTCOME_PENDING 5-35, 6-13, 7-37, F-7

backout request

- sent by BACKOUT verb 6-2

BACKOUT verb 6-2

base and optional support

- basic conversation verbs A-33
- CNOS verbs A-42
- control-operator return codes A-52
- conversation return codes A-39
- LU definition verbs A-45

base and optional support (continued)

- mapped conversation verbs A-2
- Miscellaneous COPR verbs A-51
- notes on implementation details A-53
- session control verbs A-44
- type-independent conversation verbs A-31
- what-received indications A-40
- basic conversation**
 - used as a CNOS conversation 8-2
- basic conversation verbs 7-1**
 - See also* individual verbs
 - examples of use B-1
 - for two-way alternate conversation style 2-11
 - for two-way simultaneous conversation style 2-13
- BIND_RSP_QUEUE_CAPABILITY** parameter
 - of DEFINE_LOCAL_LU verb 10-5
 - of DISPLAY_LOCAL_LU verb 10-31
- boundary, protocol**
 - See* protocol boundary
- buffering by LU**
 - general description 1-9
 - of allocation request 5-8, 7-9
 - of data 5-67, 7-68
 - of deallocation request 5-20, 7-22
 - of error notification 5-70, 7-70
 - of SEND indication 5-41, 7-43

C

change number of sessions

See CNOS

CHANGE_SESSION_LIMIT verb 8-3

changing

- (LU,mode) session limit 8-3
- contention-winner polarities 8-3
- operating parameters for a mode 10-9
- operating parameters for a remote LU 10-15
- operating parameters for a transaction program 10-20
- operating parameters for the local LU 10-3

characteristics, protocol boundary

See protocol boundary characteristics

cleanup, type of deactivation 9-4, 9-6

CNOS

- conversation 8-2
- request and reply 8-1
- transaction 8-2
- verbs 8-1

CNOS_SUPPORT parameter

- of DEFINE_REMOTE_LU verb 10-16
- of DISPLAY_REMOTE_LU verb 10-38

COMMAND_RACE_REJECT return code

for control operator verbs G-2

communication, interprogram

See interprogram communication

complete data record

- received by MC_RECEIVE_AND_WAIT verb 5-45
- received by MC_RECEIVE_IMMEDIATE verb 5-56

Compression

See also Length-checked compression

- LZ compression A-20, A-45, A-46, A-48, A-49
- RLE

See Compression, Run-length encoding

- Run-length encoding A-20, A-45, A-46, A-48, A-49

CONFIRM_DEALLOCATE indication

- received by MC_RECEIVE_AND_WAIT verb 5-46
- received by MC_RECEIVE_IMMEDIATE verb 5-57
- received by RECEIVE_AND_WAIT verb 7-47
- received by RECEIVE_IMMEDIATE verb 7-58

CONFIRM indication

- received by MC_RECEIVE_AND_WAIT verb 5-45
- received by MC_RECEIVE_IMMEDIATE verb 5-57
- received by RECEIVE_AND_WAIT verb 7-47
- received by RECEIVE_IMMEDIATE verb 7-58

CONFIRM_SEND indication

- received by MC_RECEIVE_AND_WAIT verb 5-46
- received by MC_RECEIVE_IMMEDIATE verb 5-57
- received by RECEIVE_AND_WAIT verb 7-47
- received by RECEIVE_IMMEDIATE verb 7-58

CONFIRM synchronization level 5-5, 7-5

CONFIRM verb 7-11

confirmation reply

- sent by CONFIRMED verb 7-14
- sent by MC_CONFIRMED verb 5-13

confirmation request

- received by MC_RECEIVE_AND_WAIT verb 5-45
- received by MC_RECEIVE_IMMEDIATE verb 5-57
- received by RECEIVE_AND_WAIT verb 7-47
- received by RECEIVE_IMMEDIATE verb 7-58
- sent by CONFIRM verb 7-11
- sent by MC_CONFIRM verb 5-10

CONFIRMED verb 7-14

CONLOSERS_SESSION_COUNT parameter

- of DISPLAY_MODE verb 10-35

connection, program-to-program

See program-to-program connection

contention winner and loser

See LU-LU session

contention-winner polarities

changing 8-3

contention-winner polarities (*continued*)

- initializing 8-8
- processing by target LU 8-13
- resetting 8-15

control-operator

- transaction program 7-84

control-operator verbs 7-84, 2-15

- See also* individual verbs
- correlation to return codes G-6
- subcategories 7-84

conversation

- allocated on LU-LU session 1-5
- allocation of 7-2
- changing to receive state
 - using PREPARE_TO_RECEIVE verb 7-39
 - using RECEIVE_AND_WAIT verb 7-44
- changing to send state
 - using SEND_ERROR verb 7-70
- deallocation of 7-16
- obtaining attributes of 7-30
- requesting change to send state
 - using REQUEST_TO_SEND verb 7-62
- send-receive relationship 1-9, 2-4
- starting, resource ID of 1-8
- two-way alternate conversation style 1-9, 2-4, 2-5, 2-6, 2-11
 - basic conversation verbs 2-11
 - conversation verbs supported for 2-5
 - mapped conversation verbs 2-6
- two-way simultaneous conversation style 1-9, 2-4, 2-5, 2-8, 2-13
 - basic conversation verbs 2-13
 - conversation verbs supported for 2-5
 - mapped conversation verbs 2-8

CONVERSATION_CORRELATOR parameter

- of GET_ATTRIBUTES verb 7-31
- of MC_GET_ATTRIBUTES verb 5-29

CONVERSATION_ENDED return code F-3

Conversation Group

- deactivation of
 - by DEACTIVATE_CONVERSATION_GROUP verb 9-4

CONVERSATION_GROUP_ID parameter

- of ALLOCATE verb 7-5
- of DEACTIVATE_CONVERSATION_GROUP verb 9-4
- of GET_ATTRIBUTES verb 7-31
- of MC_ALLOCATE verb 5-4
- of MC_GET_ATTRIBUTES verb 5-29

CONVERSATION_GROUP_IDS parameter

- of DISPLAY_MODE verb 10-36

conversation-level security 10-19

conversation-level security verification 10-4

conversation resource 1-4

conversation state changes

- backout-required state, entered by
 - BACKED_OUT return code F-6
 - PREPARE_FOR_SYNCPT verb 7-37
- confirm state, entered by
 - RECEIVE_AND_WAIT verb 7-48
 - RECEIVE_IMMEDIATE verb 7-59
- deallocate-pending state, entered by
 - DEALLOCATE verb 7-21
- deallocate state, entered by
 - ALLOCATION_ERROR return code F-2
 - CONFIRMED verb 7-15
 - DEALLOCATE_ABEND_PROG return code F-7
 - DEALLOCATE_ABEND return code F-7, F-13
 - DEALLOCATE_ABEND_SVC return code F-8
 - DEALLOCATE_ABEND_TIMER return code F-8
 - DEALLOCATE_NORMAL return code F-9
 - RESOURCE_FAILURE_NO_RETRY return code F-12
 - RESOURCE_FAILURE_RETRY return code F-12
 - SYNCPT verb 6-14
- defer-deallocate state, entered by
 - DEALLOCATE verb 7-21
- defer state, entered by
 - PREPARE_TO_RECEIVE verb 7-41
- receive-only state, entered by
 - DEALLOCATE verb 7-21
- receive state, entered by
 - CONFIRM verb 7-12
 - CONFIRMED verb 7-15
 - connected remote program 7-10
 - FLUSH verb 7-27
 - PREPARE_TO_RECEIVE verb 7-42
 - PROG_ERROR_PURGING return code F-11
 - RECEIVE_AND_WAIT verb 7-48
 - SVC_ERROR_PURGING return code F-12
 - SYNCPT verb 6-14
- reset state, entered by
 - ALLOCATION_ERROR return code F-13
 - DEALLOCATE_ABEND_PROG return code F-13
 - DEALLOCATE_ABEND_SVC return code F-14
 - DEALLOCATE_ABEND_TIMER return code F-14
 - DEALLOCATE verb 7-21
 - ERROR_INDICATION return code F-14
 - RECEIVE_AND_WAIT verb 7-48
 - RECEIVE_IMMEDIATE verb 7-59
 - RESOURCE_FAILURE_NO_RETRY return code F-16
 - RESOURCE_FAILURE_RETRY return code F-16

conversation state changes (*continued*)

reset state, entered by (*continued*)
 SYNCPT verb 6-14

send-only state, entered by
 DEALLOCATE_NORMAL return code F-14
 RECEIVE_AND_WAIT verb 7-48
 RECEIVE_IMMEDIATE verb 7-59

send-receive state, entered by
 ALLOCATE verb 7-8
 local and remote programs 7-10

send state, entered by
 ALLOCATE verb 7-8
 CONFIRMED verb 7-15
 MAP_EXECUTION_FAILURE return code F-9
 MAP_NOT_FOUND return code F-9
 MAPPING_NOT_SUPPORTED return code F-9
 RECEIVE_AND_WAIT verb 7-48
 RECEIVE_IMMEDIATE verb 7-58
 SEND_ERROR verb 7-73
 SYNCPT verb 6-14

state after last synchronization point, entered by
 BACKOUT verb 6-3

state unchanged by
 CONFIRM verb 7-12
 FLUSH verb 7-27
 GET_ATTRIBUTES verb 7-31
 GET_TYPE verb 6-7
 POST_ON_RECEIPT verb 7-34
 RECEIVE_AND_WAIT verb 7-48
 RECEIVE_EXPEDITED_DATA verb 7-53
 RECEIVE_IMMEDIATE verb 7-59
 REQUEST_TO_SEND verb 7-63
 SEND_DATA verb 7-68
 SEND_ERROR verb 7-73
 SEND_EXPEDITED_DATA verb 7-79
 SET_SYNCPT_OPTIONS verb 6-11
 SYNCPT verb 6-14
 TEST verb 7-81
 WAIT_FOR_COMPLETION verb 6-20
 WAIT verb 6-17

sync-point-deallocate state, entered by
 PREPARE_FOR_SYNCPT verb 7-37

sync-point-send state, entered by
 PREPARE_FOR_SYNCPT verb 7-37

sync-point state, entered by
 PREPARE_FOR_SYNCPT verb 7-37
 RECEIVE_AND_WAIT verb 7-48
 RECEIVE_IMMEDIATE verb 7-59

conversation state matrices E-1

CONVERSATION_STATE parameter
 of GET_ATTRIBUTES verb 7-31
 of MC_GET_ATTRIBUTES verb 5-29

conversation states

local program's view of E-1

conversation style 2-3

conversation type 2-3
 specified by TYPE parameter 5-3, 7-3

CONVERSATION_TYPE_MISMATCH
 See ALLOCATION_ERROR subcodes

CONVERSATION_TYPE parameter
 of DEFINE_TP verb 10-21
 of DISPLAY_TP verb 10-40

conversation verbs 4-8, 2-3
 return codes for
 See return codes for conversation verbs
 subcategories 4-8

CONWINNER_AUTO_ACTIVATE_LIMIT parameter
 of DEFINE_MODE verb 10-12
 of DISPLAY_MODE verb 10-35

CONWINNERS_SESSION_COUNT parameter
 of DISPLAY_MODE verb 10-35

CPSVCMG mode name
 on ACTIVATE verb 9-2
 on ALLOCATE verb 7-3
 on INITIALIZE_SESSION_LIMIT verb 8-8
 on RESET_SESSION_LIMIT verb 8-16

D

data

See also OK subcodes

mapping 5-65

posting receipt of 7-33

received by RECEIVE_AND_WAIT verb 7-45

received by RECEIVE_IMMEDIATE verb 7-56

sent by SEND_DATA verb 7-65

Data compression

See Length-checked compression

Data Encryption Standard (DES) 10-18

data field of a logical record 7-65

DATA_MAPPING parameter

of DEFINE_TP verb 10-25
of DISPLAY_TP verb 10-40

DATA parameter

of MC_RECEIVE_AND_WAIT verb 5-45
of MC_RECEIVE_EXPEDITED_DATA verb 5-52
of MC_RECEIVE_IMMEDIATE verb 5-56
of MC_SEND_DATA verb 5-64
of MC_SEND_EXPEDITED_DATA verb 5-76
of RECEIVE_AND_WAIT verb 7-47
of RECEIVE_EXPEDITED_DATA verb 7-53
of RECEIVE_IMMEDIATE verb 7-57
of SEND_DATA verb 7-65
of SEND_EXPEDITED_DATA verb 7-77

data record

posting receipt of 5-31

data record (continued)

- received by MC_RECEIVE_AND_WAIT verb 5-42
- received by MC_RECEIVE_IMMEDIATE verb 5-54
- sent by MC_SEND_DATA verb 5-64
- DEACTIVATE_CONVERSATION_GROUP verb 9-4
- DEACTIVATE_SESSION verb 9-6
- DEALLOCATE_ABEND
 - See ERROR_INDICATION subcodes
- DEALLOCATE_ABEND_PROG
 - See ERROR_INDICATION subcodes
- DEALLOCATE_ABEND_PROG return code F-7, F-13
- DEALLOCATE_ABEND return code F-7, F-13
- DEALLOCATE_ABEND_SVC
 - See ERROR_INDICATION subcodes
- DEALLOCATE_ABEND_SVC return code F-8, F-13
- DEALLOCATE_ABEND_TIMER
 - See ERROR_INDICATION subcodes
- DEALLOCATE_ABEND_TIMER return code F-8, F-14
- DEALLOCATE_NORMAL return code F-8, F-14
- DEALLOCATE verb 7-16
- deallocation
 - of a conversation 7-16
 - of a mapped conversation 5-15
- DEFINE_LOCAL_LU verb 10-3
- DEFINE_MODE verb 10-9
- DEFINE_REMOTE_LU verb 10-15
- DEFINE_TP verb 10-20
- DELETE verb 10-28
- deleting
 - operating parameters of local LU 10-28
- DISPLAY_LOCAL_LU verb 10-30
- DISPLAY_MODE verb 10-33
- DISPLAY_REMOTE_LU verb 10-37
- DISPLAY_SIGNED_ON_LIST verb 11-3
- DISPLAY_TP verb 10-39
- displaying
 - operating parameters for a mode 10-33
 - operating parameters for a remote LU 10-37
 - operating parameters for a transaction program 10-39
 - operating parameters for the local LU 10-30
- DRAIN_LOCAL_LU parameter
 - of DISPLAY_MODE verb 10-35
- DRAIN_REMOTE_LU parameter
 - of DISPLAY_MODE verb 10-35
- DRAIN_SOURCE parameter
 - of RESET_SESSION_LIMIT verb 8-17
- DRAIN_TARGET parameter
 - of RESET_SESSION_LIMIT verb 8-17

E

- effective program-to-program connection 1-6
- ENCRYPT parameter
 - of MC_SEND_DATA verb 5-65, 7-66
- ENCRYPTION_NOT_SUPPORTED return code F-4
- END statement 1-9
- ENTRIES_ON_FROM_LIST parameter
 - of DISPLAY_SIGNED_ON_LIST verb 11-4
- ENTRIES_ON_TO_LIST parameter
 - of DISPLAY_SIGNED_ON_LIST verb 11-4
- ERROR conditions
 - for basic conversation verbs
 - parameter check 4-5
 - for LU 6.2 verbs
 - state check 4-5
- ERROR_INDICATION return code F-14
- ERROR_INDICATION subcodes
 - for conversation verbs
 - ALLOCATION_ERROR F-14
 - DEALLOCATE_ABEND F-14
 - DEALLOCATE_ABEND_PROG F-15
 - DEALLOCATE_ABEND_SVC F-15
 - DEALLOCATE_ABEND_TIMER F-15
 - RESOURCE_FAILURE_NO_RETRY F-15
 - RESOURCE_FAILURE_RETRY F-15
 - UNKNOWN F-15
- error notification
 - indicated by return code F-10, F-16
 - sent by MC_SEND_ERROR verb 5-70
 - sent by SEND_ERROR verb 7-70
- execution
 - transaction program 1-8
 - verb 1-9
- EXPEDITED_DATA_NOT_SUPPORTED_BY_LU
 - return code F-4
- EXPEDITED_DATA_RECEIVED parameter
 - of MC_RECEIVE_EXPEDITED_DATA verb 5-52
 - of MC_SEND_EXPEDITED_DATA verb 5-77
 - of RECEIVE_EXPEDITED_DATA verb 7-53
 - of SEND_EXPEDITED_DATA verb 7-78

F

- FILL parameter
 - of POST_ON_RECEIPT verb 7-33
 - of RECEIVE_AND_WAIT verb 7-45
 - of RECEIVE_IMMEDIATE verb 7-55
- FLUSH verb 7-26
- flushing LU's send buffer
 - by ALLOCATE verb 7-9
 - by BACKOUT verb 6-2
 - by CONFIRM verb 7-11
 - by DEALLOCATE verb 7-17

flushing LU's send buffer (*continued*)

- by FLUSH verb 7-26
- by MC_ALLOCATE verb 5-8
- by MC_CONFIRM verb 5-10
- by MC_DEALLOCATE verb 5-16
- by MC_FLUSH verb 5-25
- by MC_PREPARE_FOR_SYNCPT verb 5-34
- by MC_PREPARE_TO_RECEIVE verb 5-38
- by MC_RECEIVE_AND_WAIT verb 5-42
- by MC_SEND_ERROR verb 5-70
- by PREPARE_FOR_SYNCPT verb 7-36
- by PREPARE_TO_RECEIVE verb 7-40
- by RECEIVE_AND_WAIT verb 7-44
- by SEND_ERROR verb 7-70
- by SYNCPT verb 6-12
- general description 1-9

FORCE parameter

- of RESET_SESSION_LIMIT verb 8-18

FORCED

See OK subcodes

full-duplex conversation style

See two-way simultaneous conversation style

full-duplex conversation verbs

- correlation to return codes F-21

FULL_DUPLEX_NOT_SUPPORTED_BY_LU

See ALLOCATION_ERROR subcodes

FULLY_QUALIFIED_LU_NAME parameter

- of DEFINE_LOCAL_LU verb 10-4
- of DEFINE_MODE verb 10-10
- of DEFINE_REMOTE_LU verb 10-16
- of DISPLAY_LOCAL_LU verb 10-30
- of DISPLAY_MODE verb 10-34
- of DISPLAY_REMOTE_LU verb 10-37

G

generic protocol boundary 1-3

GET_ATTRIBUTES verb 7-30

GET_TP_PROPERTIES verb 6-5

GET_TYPE verb 6-7

H

half-duplex conversation style

See two-way alternate conversation style

half-duplex conversation verbs

- correlation to return codes F-19

hex (hexadecimal) 7-65

I

INBOUND_COMPRESSION_LEVEL parameter

- of DEFINE_LOCAL_LU verb 10-5
- of DEFINE_MODE verb 10-11
- of DISPLAY_LOCAL_LU verb 10-31

INBOUND_COMPRESSION_LEVEL parameter (*continued*)

- of DISPLAY_MODE verb 10-35

incomplete data record

- received by MC_RECEIVE_AND_WAIT verb 5-45
- received by MC_RECEIVE_IMMEDIATE verb 5-56

INITIALIZE_SESSION_LIMIT verb 8-8

initializing

- (LU,mode) session limit 8-8
- contention-winner polarities 8-8
- operating parameters for a mode 10-9
- operating parameters for a remote LU 10-15
- operating parameters for a transaction program 10-20
- operating parameters for the local LU 10-3

INITIATE_TYPE parameter

- of DEFINE_REMOTE_LU verb 10-16
- of DISPLAY_REMOTE_LU verb 10-38

INSTANCE_COUNT parameter

- of DISPLAY_TP verb 10-40

INSTANCE_LIMIT parameter

- of DEFINE_TP verb 10-25
- of DISPLAY_TP verb 10-40

instance of transaction program 1-8

interconnection of programs

- initiating 1-7
- logical 1-6

interprogram communication 1-5, 1-3

INVALID_TIMEOUT_VALUE return code

issuing verbs 1-9

L

length (LL) field of a logical record 7-65

Length-checked compression 1-5, A-20, A-45, A-46, A-48, A-49

LENGTH_CHECKED_COMPRESSION parameter

- of DEFINE_LOCAL_LU verb 10-5
- of DEFINE_MODE verb 10-11
- of DISPLAY_LOCAL_LU verb 10-31
- of DISPLAY_MODE verb 10-34

LENGTH parameter

- of MC_POST_ON_RECEIPT verb 5-31
- of MC_RECEIVE_AND_WAIT verb 5-44
- of MC_RECEIVE_EXPEDITED_DATA verb 5-51
- of MC_RECEIVE_IMMEDIATE verb 5-55
- of MC_SEND_DATA verb 5-65
- of MC_SEND_EXPEDITED_DATA verb 5-76
- of POST_ON_RECEIPT verb 7-33
- of RECEIVE_AND_WAIT verb 7-45
- of RECEIVE_EXPEDITED_DATA verb 7-52
- of RECEIVE_IMMEDIATE verb 7-56

LENGTH parameter (*continued*)
 of SEND_DATA verb 7-66
 of SEND_EXPEDITED_DATA verb 7-77
 lengths of symbol strings C-1
LIST parameter
 of DISPLAY_SIGNED_ON_LIST verb 11-3
 of SIGNOFF verb 11-7
local LU
 defining operating parameters for 10-3
 definition 2-2
 deleting operating parameters of 10-28
 displaying operating parameters for 10-30
LOCAL_LU_NAME parameter
 of DELETE verb 10-28
LOCAL_MAX_SESSION_LIMIT parameter
 of DEFINE_MODE verb 10-13
 of DISPLAY_MODE verb 10-35
local program
 definition 2-2
local resources 1-1
LOCALLY_KNOWN_LU_NAME parameter
 of DEFINE_REMOTE_LU verb 10-16
 of DISPLAY_REMOTE_LU verb 10-37
LOCKS parameter
 of MC_PREPARE_TO_RECEIVE verb 5-38
 of PREPARE_TO_RECEIVE verb 7-40
LOG_DATA parameter
 of DEALLOCATE verb 7-18
 of SEND_ERROR verb 7-71
logical network 1-1
logical records
 complete and incomplete 7-68
 data field of 7-65
 length (LL) field of 7-65
 posting receipt of specifies to post when a complete
 or truncated 7-33
 received by RECEIVE_AND_WAIT verb 7-45
 received by RECEIVE_IMMEDIATE verb 7-55
 sent by SEND_DATA verb 7-65
logical resources 1-1
logical unit (LU) 1-1
See also LU
logical unit type 6.2 1-1
LU
 local 2-2
 source 8-2
LU 6.2
 protocol boundary 1-5
 type of logical unit 1-1
LU definition verbs 10-1
LU-LU password
 example for entering 10-18
 use 10-18
LU_LU_PASSWORD parameter
 of DEFINE_REMOTE_LU verb 10-16

LU-LU session
 activation
 by ACTIVATE_SESSION verb 9-2
 by CHANGE_SESSION_LIMIT verb 8-3
 by INITIALIZE_SESSION_LIMIT verb 8-8
 allocation of 5-2, 7-2
 contention loser 1-5
 contention winner 1-5
 conversation allocated on 1-5
 deactivation
 by CHANGE_SESSION_LIMIT verb 8-3
 by DEACTIVATE_SESSION verb 9-6
 by RESET_SESSION_LIMIT verb 8-15
 logical connection 1-1
 LUs connected by 1-5
 serially reusable resource 1-4
LU_MODE_SESSION_COUNT parameter
 of DISPLAY_MODE verb 10-35
LU_MODE_SESSION_LIMIT_CLOSED return code
 for control operator verbs G-2
LU_MODE_SESSION_LIMIT_EXCEEDED return code
 for control operator verbs G-2
LU_MODE_SESSION_LIMIT_NOT_ZERO return code
 for control operator verbs G-2
LU_MODE_SESSION_LIMIT parameter
 of CHANGE_SESSION_LIMIT verb 8-3
 of DISPLAY_MODE verb 10-35
 of INITIALIZE_SESSION_LIMIT verb 8-9
LU_MODE_SESSION_LIMIT_ZERO return code
 for control operator verbs G-3
LU_NAME parameter
 of ACTIVATE_SESSION verb 9-2
 of ALLOCATE verb 7-3
 of CHANGE_SESSION_LIMIT verb 8-3
 of INITIALIZE_SESSION_LIMIT verb 8-8
 of MC_ALLOCATE verb 5-3
 of PROCESS_SESSION_LIMIT verb 8-13
 of RESET_SESSION_LIMIT verb 8-15
LU parameter
 of SIGNOFF verb 11-8
LU services component program 7-1
LU_SESSION_COUNT parameter
 of DISPLAY_LOCAL_LU verb 10-31
LU_SESSION_LIMIT_EXCEEDED return code
 for control operator verbs G-3
LU_SESSION_LIMIT parameter
 of DEFINE_LOCAL_LU verb 10-4
 of DISPLAY_LOCAL_LU verb 10-30
LU,mode session limit
 changing 8-3
 initializing 8-8
 processing by target LU 8-13
 resetting 8-15

LU,mode session limit (*continued*)
 specified by LU_MODE_SESSION_LIMIT parameter 8-3
LUW_IDENTIFIER parameter
 of GET_TP_PROPERTIES verb 6-6
LUW_OUTCOME_MIXED
See BACKED_OUT subcodes
See OK subcodes
LUW_OUTCOME_PENDING
See BACKED_OUT subcodes
See OK subcodes
LZ compression
See Compression, LZ compression

M

MAP_EXECUTION_FAILURE return code F-9, F-15
MAP_NAME parameter
 of DEFINE_LOCAL_LU verb 10-5
 of MC_RECEIVE_AND_WAIT verb 5-46
 of MC_RECEIVE_IMMEDIATE verb 5-57
 of MC_SEND_DATA verb 5-65
MAP_NAMES parameter
 of DISPLAY_LOCAL_LU verb 10-31
MAP_NOT_FOUND return code F-9, F-15
mapped conversation
 allocation of 5-2
 changing to receive state
 using MC_PREPARE_TO_RECEIVE verb 5-37
 using MC_RECEIVE_AND_WAIT verb 5-42
 changing to send state
 using MC_SEND_ERROR verb 5-70
 deallocation of 5-15
 obtaining attributes of 5-28
 requesting change to send state
 using MC_REQUEST_TO_SEND verb 5-61
mapped conversation state changes
 backout-required state, entered by
 MC_PREPARE_FOR_SYNCPT verb 5-35
 confirm state, entered by
 MC_RECEIVE_AND_WAIT verb 5-47
 MC_RECEIVE_IMMEDIATE verb 5-58
 deallocate-pending state, entered by
 MC_DEALLOCATE verb 5-20
 deallocate state, entered by
 MC_CONFIRMED verb 5-14
 defer-deallocate state, entered by
 MC_DEALLOCATE verb 5-20
 defer-receive state, entered by
 MC_PREPARE_TO_RECEIVE verb 5-40
 receive-only state, entered by
 MC_DEALLOCATE verb 5-20
 receive state, entered by
 connected remote program 5-9
 MC_CONFIRM verb 5-11
 MC_CONFIRMED verb 5-14

mapped conversation state changes (*continued*)
 receive state, entered by (*continued*)
 MC_FLUSH verb 5-26
 MC_PREPARE_TO_RECEIVE verb 5-40
 MC_RECEIVE_AND_WAIT verb 5-47
 reset state, entered by
 MC_DEALLOCATE verb 5-20
 MC_RECEIVE_AND_WAIT verb 5-47
 MC_RECEIVE_IMMEDIATE verb 5-58
 send-only state, entered by
 MC_RECEIVE_AND_WAIT verb 5-47
 MC_RECEIVE_IMMEDIATE verb 5-58
 send-receive state, entered by
 local and remote programs 5-9
 MC_ALLOCATE verb 5-8
 send state, entered by
 MC_ALLOCATE verb 5-8
 MC_CONFIRMED verb 5-14
 MC_RECEIVE_AND_WAIT verb 5-47
 MC_RECEIVE_IMMEDIATE verb 5-58
 MC_SEND_ERROR verb 5-72
 state unchanged by
 GET_TP_PROPERTIES verb 6-6
 MC_CONFIRM verb 5-11
 MC_FLUSH verb 5-26
 MC_GET_ATTRIBUTES verb 5-29
 MC_POST_ON_RECEIPT verb 5-31
 MC_RECEIVE_AND_WAIT verb 5-47
 MC_RECEIVE_EXPEDITED_DATA verb 5-52
 MC_RECEIVE_IMMEDIATE verb 5-58
 MC_REQUEST_TO_SEND verb 5-62
 MC_SEND_DATA verb 5-67
 MC_SEND_ERROR verb 5-72
 MC_SEND_EXPEDITED_DATA verb 5-78
 MC_TEST verb 5-80
 sync-point-deallocate state, entered by
 MC_PREPARE_FOR_SYNCPT verb 5-35
 sync-point-send state, entered by
 MC_PREPARE_FOR_SYNCPT verb 5-35
 sync-point state, entered by
 MC_PREPARE_FOR_SYNCPT verb 5-35
 MC_RECEIVE_AND_WAIT verb 5-47
 MC_RECEIVE_IMMEDIATE verb 5-58
mapped conversation verbs 5-1
See also individual verbs
 for two-way alternate conversation style 2-6
 for two-way simultaneous conversation style 2-8
MAPPING_NOT_SUPPORTED return code F-9, F-16
mapping of data 5-65
 MC_ALLOCATE verb 5-2
 MC_CONFIRM verb 5-10
 MC_CONFIRMED verb 5-13

MC_DEALLOCATE verb 5-15
MC_FLUSH verb 5-25
MC_GET_ATTRIBUTES verb 5-28
MC_POST_ON_RECEIPT verb 5-31
MC_PREPARE_FOR_SYNCPT verb 5-34
MC_PREPARE_TO_RECEIVE verb 5-37
MC_RECEIVE_AND_WAIT verb 5-42
MC_RECEIVE_EXPEDITED_DATA verb 5-50
MC_RECEIVE_IMMEDIATE verb 5-54
MC_REQUEST_TO_SEND verb 5-61
 attention mechanism
 See **MC_REQUEST_TO_SEND** verb
MC_SEND_DATA verb 5-64
MC_SEND_ERROR verb 5-70
MC_SEND_EXPEDITED_DATA verb 5-76
MC_TEST verb 5-79
MIN_CONLOSERS parameter
 of **DISPLAY_MODE** verb 10-35
MIN_CONWINNERS parameter
 of **DISPLAY_MODE** verb 10-35
MIN_CONWINNERS_SOURCE parameter
 of **CHANGE_SESSION_LIMIT** verb 8-4
 of **INITIALIZE_SESSION_LIMIT** verb 8-9
MIN_CONWINNERS_TARGET parameter
 of **CHANGE_SESSION_LIMIT** verb 8-4
 of **INITIALIZE_SESSION_LIMIT** verb 8-9
Miscellaneous COPS verbs 11-1
mode
 defining operating parameters for 10-9
 displaying operating parameters for 10-33
MODE_NAME parameter
 of **ACTIVATE_SESSION** verb 9-2
 of **ALLOCATE** verb 7-3
 of **CHANGE_SESSION_LIMIT** verb 8-3
 of **DEFINE_MODE** verb 10-10
 of **DELETE** verb 10-28
 of **DISPLAY_MODE** verb 10-34
 of **GET_ATTRIBUTES** verb 7-31
 of **INITIALIZE_SESSION_LIMIT** verb 8-8
 of **MC_ALLOCATE** verb 5-3
 of **MC_GET_ATTRIBUTES** verb 5-29
 of **PROCESS_SESSION_LIMIT** verb 8-13
 of **RESET_SESSION_LIMIT** verb 8-16
MODE_NAMES parameter
 of **DISPLAY_REMOTE_LU** verb 10-38

N

name
 LU 5-3, 7-3
 mode 5-3, 7-3
 transaction program 1-8, 5-3, 7-3
negotiation of CNOS parameters G-3
network
 logical 1-1

network (*continued*)
 physical 1-1
network properties designated by mode name 5-3, 7-3
NO_LU_VERIFICATION subparameter
 of **PIP** parameter 10-25
nonblocking support 3-9
NONE access security 5-5, 7-5
NONE synchronization level 5-5
NONE synchronization level specifies that the programs
 will not perform 7-5
normal, type of deactivation 9-4, 9-6
NOT_DATA
 See **OK** subcodes

O

OK return code
 for control operator verbs G-3
 for conversation verbs F-4, F-9
OK subcodes
 for control operator verbs
 AS_NEGOTIATED G-3
 AS_SPECIFIED G-3
 FORCED G-3
 for conversation verbs
 DATA F-4, F-9
 NOT_DATA F-9
 for sync-point verbs
 ALL_AGREED 6-13, F-10
 LUW_OUTCOME_MIXED 6-13, F-10
 LUW_OUTCOME_PENDING 6-13, F-10
 VOTED_READ_ONLY 6-13, F-10
OK_TO_LEAVE_OUT parameter
 of **SET_SYNCPT_OPTIONS** verb 6-9
operating parameters for a mode
 defining 10-9
 displaying 10-33
operating parameters for a remote LU
 defining 10-15
 displaying 10-37
operating parameters for a transaction program
 defining 10-20
 displaying 10-39
operating parameters for the local LU
 defining 10-3
 displaying 10-30
operating parameters of local LU
 deleting 10-28
OPERATION_INCOMPLETE return code
 for conversation verbs F-4
OPERATION_NOT_ACCEPTED return code
 for conversation verbs F-4
OUTBOUND_COMPRESSION_LEVEL parameter
 of **DEFINE_LOCAL_LU** verb 10-6
 of **DEFINE_MODE** verb 10-12

OUTBOUND_COMPRESSION_LEVEL parameter
(continued)

of **DISPLAY_LOCAL_LU** verb 10-31
of **DISPLAY_MODE** verb 10-35

overview description of verbs 2-1

OWN_FULLY_QUALIFIED_LU_NAME parameter

of **GET_TP_PROPERTIES** verb 6-5

OWN_TP_INSTANCE parameter

of **GET_TP_PROPERTIES** verb 6-6

OWN_TP_NAME parameter

of **GET_TP_PROPERTIES** verb 6-5

P

PARALLEL_SESSION_SUPPORT parameter

of **DEFINE_REMOTE_LU** verb 10-16

of **DISPLAY_REMOTE_LU** verb 10-38

parameter

arguments

keyword 4-2

variable 4-2

vertical list of 4-3

name 4-2

parameters

parameter check

See also **ERROR** conditions

return codes indicating F-5

PARAMETER_ERROR return code

for control operator verbs G-3

for conversation verbs F-4

parameters, verb

See parameter, parameters

PARTNER_FULLY_QUALIFIED_LU_NAME parameter

of **GET_ATTRIBUTES** verb 7-31

of **MC_GET_ATTRIBUTES** verb 5-29

PARTNER_LU_NAME parameter

of **GET_ATTRIBUTES** verb 7-31

of **MC_GET_ATTRIBUTES** verb 5-29

persistent verification 10-17

PGM access security 5-5, 7-5

physical network 1-1

PIP

See program initialization parameters (PIP)

PIP_NOT_ALLOWED

See **ALLOCATION_ERROR** subcodes

PIP_NOT_SPECIFIED_CORRECTLY

See **ALLOCATION_ERROR** subcodes

PIP parameter

of **ALLOCATE** verb 7-6

of **DEFINE_TP** verb 10-25

of **DISPLAY_TP** verb 10-40

of **MC_ALLOCATE** verb 5-6

POST_ON_RECEIPT verb 7-33

POSTING_NOT_ACTIVE return code F-10

PREFERRED_RECEIVE_RU_SIZE parameter

of **DEFINE_MODE** verb 10-12

of **DISPLAY_MODE** verb 10-36

PREFERRED_SEND_RU_SIZE parameter

of **DEFINE_MODE** verb 10-12

of **DISPLAY_MODE** verb 10-36

PREPARE_FOR_SYNCPT verb 7-36

PREPARE_TO_RECEIVE verb 7-39

PRIVILEGE parameter

of **DEFINE_TP** verb 10-25

of **DISPLAY_TP** verb 10-40

PROCEDURE statement 1-8

PROCESS_SESSION_LIMIT verb 8-13

PROCESS_SIGN_OFF verb 11-5

processing by target LU

(LU,mode) session limit 8-13

contention-winner polarities 8-13

PROFILE parameter

of **SIGNOFF** verb 11-8

PROG_ERROR_NO_TRUNC return code F-10, F-16

PROG_ERROR_PURGING return code F-11

PROG_ERROR_TRUNC return code F-11, F-16

program

interconnection 1-6

local 2-2

remote 2-2

transaction

See transaction program

program initialization parameters (PIP)

on **ALLOCATE** verb 7-6

on **MC_ALLOCATE** verb 5-6

on **PROCEDURE** statement 1-9

PROGRAM_PARAMETER_CHECK return code F-5

PROGRAM_STATE_CHECK return code F-5

program-to-program connection

effective 1-6

through SNA network 1-6

PROTECTED_LUW_IDENTIFIER parameter

of **GET_TP_PROPERTIES** verb 6-6

protected resource

See also resources, protected

allocating a conversation as 7-5

allocating a mapped conversation as 5-5

backout of 6-2

sync point of 5-34, 6-12, 7-36

protocol boundary

definition 1-3

generic 1-3

LU 6.2 1-5

states 1-7

structure 1-7

verbs 1-7

protocol boundary characteristics

attention mechanism 1-4

protocol boundary characteristics (*continued*)

- commitment control 1-4
- conversation lifetime 1-4
- conversation overhead 1-4
- efficient allocation 1-4
- error notification 1-4
- levels of conversations 1-5
- mode of service 1-5
- simultaneous activation 1-4
- subset definition 1-5
- symmetry 1-4
- sync-point service 1-4
- two-way alternate conversation style 1-4
- two-way simultaneous conversation style 1-4

purging of information

- by MC_SEND_ERROR verb 5-73
- by SEND_ERROR verb 7-74
- return codes indicating F-11

R

RECEIVE_AND_WAIT verb 7-44

receive buffer of LU

- accumulating data in 1-10

RECEIVE_EXPEDITED_DATA verb 7-51

RECEIVE_IMMEDIATE verb 7-55

RECEIVE_MAX_RU_SIZE_LOWER_BOUND parameter

- of DEFINE_MODE verb 10-10
- of DISPLAY_MODE verb 10-34

RECEIVE_MAX_RU_SIZE_UPPER_BOUND parameter

- of DEFINE_MODE verb 10-10
- of DISPLAY_MODE verb 10-34

receive state

- changing conversation to

- using PREPARE_TO_RECEIVE verb 7-39

- using RECEIVE_AND_WAIT verb 7-44

- changing mapped conversation to

- using MC_PREPARE_TO_RECEIVE verb 5-37

- using MC_RECEIVE_AND_WAIT verb 5-42

receive support of symbol strings C-3

receiving information

- using MC_RECEIVE_AND_WAIT verb 5-42

- using MC_RECEIVE_EXPEDITED_DATA verb 5-50

- using MC_RECEIVE_IMMEDIATE verb 5-54

- using RECEIVE_AND_WAIT verb 7-44

- using RECEIVE_EXPEDITED_DATA verb 7-51

- using RECEIVE_IMMEDIATE verb 7-55

remote LU

- defining operating parameters for 10-15

- definition 2-2

- displaying operating parameters for 10-37

- specified by LU_NAME parameter 5-3, 7-3

REMOTE_LU_NAME parameter

- of DELETE verb 10-28

REMOTE_LU_NAMES parameter

- of DISPLAY_LOCAL_LU verb 10-31

remote program

- definition 2-2

- specified by TPN parameter 5-3, 7-3

remote resources 1-1

removing

- users from signed-on lists 11-7

REQUEST_EXCEEDS_MAX_ALLOWED return code

- for control operator verbs G-3

REQUEST_TO_SEND_RECEIVED parameter

- of CONFIRM verb 7-12

- of MC_CONFIRM verb 5-11

- of MC_RECEIVE_AND_WAIT verb 5-44

- of MC_RECEIVE_EXPEDITED_DATA verb 5-52

- of MC_RECEIVE_IMMEDIATE verb 5-56

- of MC_SEND_DATA verb 5-66

- of MC_SEND_ERROR verb 5-72

- of MC_SEND_EXPEDITED_DATA verb 5-77

- of RECEIVE_AND_WAIT verb 7-46

- of RECEIVE_EXPEDITED_DATA verb 7-53

- of RECEIVE_IMMEDIATE verb 7-57

- of SEND_DATA verb 7-67

- of SEND_ERROR verb 7-72

- of SEND_EXPEDITED_DATA verb 7-78

REQUEST_TO_SEND verb 7-62

RESET_SESSION_LIMIT verb 8-15

resetting

- (LU,mode) session limit 8-15

- contention-winner polarities 8-15

resource

- obtaining type of 6-7

resource access

RESOURCE_FAILURE_NO_RETRY

- See ERROR_INDICATION subcodes

RESOURCE_FAILURE_NO_RETRY return code

- for control operator verbs G-4

- for conversation verbs F-11, F-16

RESOURCE_FAILURE_RETRY

- See ERROR_INDICATION subcodes

RESOURCE_FAILURE_RETRY return code

- for control operator verbs G-4

- for conversation verbs F-12, F-16

resource ID

- assigned to conversation 7-2

- assigned to mapped conversation 5-2

- of starting conversation 1-8

- unassigned from conversation 7-16

- unassigned from mapped conversation 5-15

RESOURCE_LIST parameter

- of WAIT verb 6-16

RESOURCE parameter

- of ALLOCATE verb 7-7
- of CONFIRM verb 7-11
- of CONFIRMED verb 7-14
- of DEALLOCATE verb 7-17
- of FLUSH verb 7-26
- of GET_ATTRIBUTES verb 7-30
- of GET_TYPE verb 6-7
- of MC_ALLOCATE verb 5-8
- of MC_CONFIRM verb 5-10
- of MC_CONFIRMED verb 5-13
- of MC_DEALLOCATE verb 5-16
- of MC_FLUSH verb 5-25
- of MC_GET_ATTRIBUTES verb 5-28
- of MC_POST_ON_RECEIPT verb 5-31
- of MC_PREPARE_FOR_SYNCPT verb 5-34
- of MC_PREPARE_TO_RECEIVE verb 5-37
- of MC_RECEIVE_AND_WAIT verb 5-43
- of MC_RECEIVE_EXPEDITED_DATA verb 5-50
- of MC_RECEIVE_IMMEDIATE verb 5-54
- of MC_REQUEST_TO_SEND verb 5-61
- of MC_SEND_DATA verb 5-64
- of MC_SEND_ERROR verb 5-70
- of MC_SEND_EXPEDITED_DATA verb 5-76
- of MC_TEST verb 5-79
- of POST_ON_RECEIPT verb 7-33
- of PREPARE_FOR_SYNCPT verb 7-36
- of PREPARE_TO_RECEIVE verb 7-39
- of PROCESS_SESSION_LIMIT verb 8-13
- of PROCESS_SIGN_OFF verb 11-5
- of RECEIVE_AND_WAIT verb 7-45
- of RECEIVE_EXPEDITED_DATA verb 7-51
- of RECEIVE_IMMEDIATE verb 7-55
- of REQUEST_TO_SEND verb 7-62
- of SEND_DATA verb 7-65
- of SEND_ERROR verb 7-70
- of SEND_EXPEDITED_DATA verb 7-77
- of TEST verb 7-80

RESOURCE_POSTED parameter

- of WAIT verb 6-17

resources

conversation

See conversation resource

examples of 1-1

local 1-1

logical 1-1

protected 1-4

See also protected resource

remote 1-1

serially reusable 1-4

state representation of 1-1

unprotected 1-4, 6-15

RESPONSIBLE parameter

- of CHANGE_SESSION_LIMIT verb 8-4

RESPONSIBLE parameter (continued)

- of RESET_SESSION_LIMIT verb 8-16

retrieving

- signed on list information 11-3

RETURN_CODE parameter

- of ACTIVATE_SESSION verb 9-2
- of ALLOCATE verb 7-7
- of BACKOUT verb 6-3
- of CHANGE_SESSION_LIMIT verb 8-4
- of CONFIRM verb 7-12
- of DEACTIVATE_CONVERSATION_GROUP verb 9-5
- of DEACTIVATE_SESSION verb 9-7
- of DEALLOCATE verb 7-18
- of DEFINE_LOCAL_LU verb 10-6
- of DEFINE_MODE verb 10-13
- of DEFINE_REMOTE_LU verb 10-17
- of DEFINE_TP verb 10-26
- of DELETE verb 10-29
- of DISPLAY_LOCAL_LU verb 10-30
- of DISPLAY_MODE verb 10-34
- of DISPLAY_REMOTE_LU verb 10-37
- of DISPLAY_SIGNED_ON_LIST verb 11-3, 11-8
- of DISPLAY_TP verb 10-39
- of FLUSH verb 7-27
- of GET_ATTRIBUTES verb 7-30
- of GET_TP_PROPERTIES verb 6-5
- of GET_TYPE verb 6-7, 7-15
- of INITIALIZE_SESSION_LIMIT verb 8-10
- of MC_ALLOCATE verb 5-6
- of MC_CONFIRM verb 5-11
- of MC_CONFIRMED verb 5-14
- of MC_DEALLOCATE verb 5-17
- of MC_FLUSH verb 5-26
- of MC_GET_ATTRIBUTES verb 5-28
- of MC_POST_ON_RECEIPT verb 5-31
- of MC_PREPARE_FOR_SYNCPT verb 5-35
- of MC_PREPARE_TO_RECEIVE verb 5-39
- of MC_RECEIVE_AND_WAIT verb 5-44
- of MC_RECEIVE_EXPEDITED_DATA verb 5-51
- of MC_RECEIVE_IMMEDIATE verb 5-55
- of MC_REQUEST_TO_SEND verb 5-61
- of MC_SEND_DATA verb 5-66
- of MC_SEND_ERROR verb 5-71
- of MC_SEND_EXPEDITED_DATA verb 5-77
- of MC_TEST verb 5-79
- of POST_ON_RECEIPT verb 7-34
- of PREPARE_FOR_SYNCPT verb 7-37
- of PREPARE_TO_RECEIVE verb 7-41
- of PROCESS_SESSION_LIMIT verb 8-13
- of PROCESS_SIGN_OFF verb 11-5
- of RECEIVE_AND_WAIT verb 7-46
- of RECEIVE_EXPEDITED_DATA verb 7-52
- of RECEIVE_IMMEDIATE verb 7-56

RETURN_CODE parameter (*continued*)

of REQUEST_TO_SEND verb 7-63
 of RESET_SESSION_LIMIT verb 8-18
 of SEND_DATA verb 7-66
 of SEND_ERROR verb 7-71
 of SEND_EXPEDITED_DATA verb 7-78
 of SET_SYNCPT_OPTIONS verb 6-10
 of SYNCPT verb 6-12
 of TEST verb 7-80
 of WAIT_FOR_COMPLETION verb 6-19
 of WAIT verb 6-16

return codes for control operator verbs

ACTIVATION_FAILURE_NO_RETRY G-1
 ACTIVATION_FAILURE_RETRY G-1
 ALLOCATION_ERROR G-1
 ALLOCATION_ERROR subcodes
 See ALLOCATION_ERROR subcodes
 COMMAND_REJECT G-2
 correlation table G-6
 LU_MODE_SESSION_LIMIT_CLOSED G-2
 LU_MODE_SESSION_LIMIT_EXCEEDED G-2
 LU_MODE_SESSION_LIMIT_NOT_ZERO G-2
 LU_MODE_SESSION_LIMIT_ZERO G-3
 LU_SESSION_LIMIT_EXCEEDED G-3
 OK G-3
 PARAMETER_ERROR G-3
 REQUEST_EXCEEDS_MAX_ALLOWED G-3
 RESOURCE_FAILURE_NO_RETRY G-4
 RESOURCE_FAILURE_RETRY G-4
 UNRECOGNIZED_MODE_NAME G-4

return codes for conversation verbs

ALL_UNKNOWN_WAIT_OBJECTS F-2
 ALLOCATION_ERROR F-2, F-6, F-13
 ALLOCATION_ERROR subcodes
 See ALLOCATION_ERROR subcodes
 BACKED_OUT F-6
 CONVERSATION_ENDED F-3
 DEALLOCATE_ABEND F-7, F-13
 DEALLOCATE_ABEND_PROG F-7, F-13
 DEALLOCATE_ABEND_SVC F-8, F-13
 DEALLOCATE_ABEND_TIMER F-8, F-14
 DEALLOCATE_NORMAL F-8, F-14
 ENCRYPTION_NOT_SUPPORTED F-4
 ERROR_INDICATION F-14
 ERROR_INDICATION subcodes
 See ERROR_INDICATION subcodes
 EXPEDITED_DATA_NOT_SUPPORTED_BY_LU F-4
 INVALID_TIMEOUT_VALUE F-4
 MAP_EXECUTION_FAILURE F-9, F-15
 MAP_NOT_FOUND F-9, F-15
 MAPPING_NOT_SUPPORTED F-9, F-16
 OK F-4, F-9
 OK subcodes
 See OK subcodes
 OPERATION_INCOMPLETE F-4

return codes for conversation verbs (*continued*)

OPERATION_NOT_ACCEPTED F-4
 PARAMETER_ERROR F-4
 POSTING_NOT_ACTIVE F-10
 PROG_ERROR_NO_TRUNC F-10, F-16
 PROG_ERROR_PURGING F-11
 PROG_ERROR_TRUNC F-11, F-16
 PROGRAM_PARAMETER_CHECK F-5
 PROGRAM_STATE_CHECK F-5
 RESOURCE_FAILURE_NO_RETRY F-11, F-16
 RESOURCE_FAILURE_RETRY F-12, F-16
 SVC_ERROR_NO_TRUNC F-12, F-17
 SVC_ERROR_PURGING F-12
 SVC_ERROR_TRUNC F-12, F-17
 TIMER_EXPIRED F-5
 UNSUCCESSFUL F-5
 USER_CONTROL_DATA_NOT_SUPPORTED F-5

return codes for full-duplex conversation verbs

correlation table F-21

return codes for half-duplex conversation verbs

correlation table F-19

RETURN_CONTROL parameter

of ALLOCATE verb 7-3
 of DEACTIVATE_CONVERSATION_GROUP verb 9-5
 of DEACTIVATE_SESSION verb 9-7
 of MC_ALLOCATE verb 5-3
 of MC_RECEIVE_EXPEDITED_DATA verb 5-50
 of RECEIVE_EXPEDITED_DATA verb 7-51

RETURN statement 1-9

returned parameters 4-3

RLE

See Compression, Run-length encoding

RLE_BEFORE_LZ parameter

of DEFINE_LOCAL_LU verb 10-6
 of DEFINE_MODE verb 10-12
 of DISPLAY_LOCAL_LU verb 10-31
 of DISPLAY_MODE verb 10-35

Run-length encoding

See Compression, Run-length encoding

S

SAME access security 5-5, 7-5
 security 11-1

conversation-level 10-19

Persistent Verification 11-1

session-level 10-18

verification, conversation-level 10-4

SECURITY_ACCEPTANCE_LOCAL_LU parameter

of DISPLAY_REMOTE_LU verb 10-38

SECURITY_ACCEPTANCE parameter

of DEFINE_REMOTE_LU verb 10-17

SECURITY_ACCEPTANCE_REMOTE_LU parameter
of **DISPLAY_REMOTE_LU** verb 10-38

SECURITY_ACCESS parameter
of **DEFINE_TP** verb 10-23
of **DISPLAY_TP** verb 10-40

SECURITY_NOT_VALID
See **ALLOCATION_ERROR** subcodes

SECURITY parameter
of **ALLOCATE** verb 7-5
of **DEFINE_LOCAL_LU** verb 10-4
of **DISPLAY_LOCAL_LU** verb 10-31
of **MC_ALLOCATE** verb 5-5

SECURITY_PROFILE parameter
of **GET_TP_PROPERTIES** verb 6-6

SECURITY_REQUIRED parameter
of **DEFINE_TP** verb 10-22
of **DISPLAY_TP** verb 10-40

SECURITY_USER_ID parameter
of **GET_TP_PROPERTIES** verb 6-6

SELECT_LAST_AGENT parameter
of **SET_SYNCPT_OPTIONS** verb 6-10

send buffer of LU
accumulating data in 1-9
See also buffering by LU
flushing 1-9
See also flushing LU's send buffer

SEND_DATA verb 7-65

SEND_ERROR verb 7-70

SEND_EXPEDITED_DATA verb 7-77

SEND indication
received by **MC_RECEIVE_AND_WAIT**
verb 5-45
received by **MC_RECEIVE_IMMEDIATE**
verb 5-57
received by **RECEIVE_AND_WAIT** verb 7-47
received by **RECEIVE_IMMEDIATE** verb 7-58
sent by **MC_PREPARE_TO_RECEIVE** verb 5-40
sent by **MC_RECEIVE_AND_WAIT** verb 5-42
sent by **PREPARE_TO_RECEIVE** verb 7-42
sent by **RECEIVE_AND_WAIT** verb 7-44

SEND_MAX_RU_SIZE_LOWER_BOUND parameter
of **DEFINE_MODE** verb 10-10
of **DISPLAY_MODE** verb 10-34

SEND_MAX_RU_SIZE_UPPER_BOUND parameter
of **DEFINE_MODE** verb 10-10
of **DISPLAY_MODE** verb 10-34

send state
changing conversation to
using **SEND_ERROR** verb 7-70
changing mapped conversation to
using **MC_SEND_ERROR** verb 5-70
requesting change in conversation to
using **REQUEST_TO_SEND** verb 7-62
requesting change in mapped conversation to
using **MC_REQUEST_TO_SEND** verb 5-61

send support of symbol strings C-3

sending data
using **MC_SEND_DATA** verb 5-64
using **SEND_DATA** verb 7-65

sending error notification
using **MC_SEND_ERROR** verb 5-70
using **SEND_ERROR** verb 7-70

sending expedited data
using **MC_SEND_EXPEDITED_DATA** verb 5-76
using **SEND_EXPEDITED_DATA** verb 7-77

SENSE_CODE parameter
of **DEACTIVATE_CONVERSATION_GROUP**
verb 9-4
of **DEACTIVATE_SESSION** verb 9-6

serially reusable resource 1-4

service TP
See SNA service transaction program

session
See LU-LU session

session control verbs 9-1

SESSION_DEACTIVATED_TP_NAME parameter
of **DEFINE_MODE** verb 10-13

SESSION_DEACTIVATEDTP_NAME parameter
of **DISPLAY_MODE** verb 10-36

SESSION_ID parameter
of **DEACTIVATE_SESSION** verb 9-6
of **GET_ATTRIBUTES** verb 7-31
of **MC_GET_ATTRIBUTES** verb 5-29

SESSION_IDS parameter
of **DISPLAY_MODE** verb 10-35

SESSION_LEVEL_CRYPTOGRAPHY parameter
of **DEFINE_MODE** verb 10-11
of **DISPLAY_MODE** verb 10-34

session-level security 10-18

SET_SYNCPT_OPTIONS verb 6-9

SIGNED_ON_FROM_LIST parameter
of **DISPLAY_SIGNED_ON_LIST** verb 11-4

SIGNED_ON_TO_LIST parameter
of **DISPLAY_SIGNED_ON_LIST** verb 11-4

SIGNOFF verb 11-7

SINGLE_SESSION_REINITIATION parameter
of **DEFINE_MODE** verb 10-10
of **DISPLAY_MODE** verb 10-34

SNA
See Systems Network Architecture (SNA)

SNA service transaction program 7-1, D-1
CNOS service transaction program 8-1, 8-14
specified by **TPN** parameter 7-3

SNASVCMG mode name
for CNOS conversation 8-2
on **ACTIVATE** verb 9-2
on **ALLOCATE** verb 7-3
on **INITIALIZE_SESSION_LIMIT** verb 8-8
on **RESET_SESSION_LIMIT** verb 8-16

- source LU
 - definition 8-2
 - returned on LU_NAME parameter 8-13
 - source
 - See source LU
 - target 8-2
- specification of symbol strings C-3
- starting
 - sign off TP processing 11-5
- state changes
 - conversation
 - See conversation state changes
- state check
 - See also ERROR conditions
 - return codes indicating F-5
- state representation of resources 1-1
- states
 - conversation
 - See conversation states
- STATUS_LIST parameter
 - of WAIT_FOR_COMPLETION verb 6-20
- STATUS parameter
 - of DEFINE_TP verb 10-21
 - of DISPLAY_TP verb 10-39
- structure
 - protocol boundary 1-7
 - transaction program 1-8
- supplied-and-returned parameters 4-3
- supplied parameters 4-3
- SVC_ERROR_NO_TRUNC return code F-12, F-17
- SVC_ERROR_PURGING return code F-12
- SVC_ERROR_TRUNC return code F-12, F-17
- symbol string
 - conventions C-1
 - length C-1
 - type C-1
- symbol string processing C-5, C-6
- SYNC_LEVEL_NOT_SUPPORTED_BY_LU
 - See ALLOCATION_ERROR subcodes
- SYNC_LEVEL_NOT_SUPPORTED_BY_PGM
 - See ALLOCATION_ERROR subcodes
- SYNC_LEVEL parameter
 - of ALLOCATE verb 7-5
 - of DEFINE_TP verb 10-22
 - of DISPLAY_TP verb 10-40
 - of GET_ATTRIBUTES verb 7-31
 - of MC_ALLOCATE verb 5-5
 - of MC_GET_ATTRIBUTES verb 5-29
- sync point options
 - set by SET_SYNCPT_OPTIONS verb 6-9
- sync-point request
 - received by MC_RECEIVE_AND_WAIT verb 5-46
 - received by RECEIVE_AND_WAIT verb 7-47
 - received by RECEIVE_IMMEDIATE verb 7-58

- sync-point request (*continued*)
 - sent by MC_PREPARE_FOR_SYNCPT verb 5-34
 - sent by PREPARE_FOR_SYNCPT verb 7-36
 - sent by SYNCPT verb 6-12
- sync-point service 1-4
- synchronization level
 - changing conversation to receive state based on 7-39
 - changing mapped conversation to receive state based on specifies to 5-37
 - deallocating conversation based on specifies to perform deallocation 7-17
 - deallocating mapped conversation based on 5-16
 - for conversation 7-5
 - for mapped conversation 5-5
- SYNCPT synchronization level 5-5, 7-5
- SYNCPT verb 6-12
- Systems Network Architecture (SNA) 1-1

T

- TAKE_SYNCPT_DEALLOCATE indication
 - received by MC_RECEIVE_AND_WAIT verb 5-46
 - received by MC_RECEIVE_IMMEDIATE verb 5-57
 - received by RECEIVE_AND_WAIT verb 7-48
 - received by RECEIVE_IMMEDIATE verb 7-58
- TAKE_SYNCPT indication
 - received by MC_RECEIVE_AND_WAIT verb 5-46
 - received by MC_RECEIVE_IMMEDIATE verb 5-57
 - received by RECEIVE_AND_WAIT verb 7-47
 - received by RECEIVE_IMMEDIATE verb 7-58
- TAKE_SYNCPT_SEND indication
 - received by MC_RECEIVE_AND_WAIT verb 5-46
 - received by MC_RECEIVE_IMMEDIATE verb indicates the remote program has 5-57
 - received by RECEIVE_AND_WAIT verb 7-48
 - received by RECEIVE_IMMEDIATE verb 7-58
- target LU
 - definition 8-2
 - specified by LU_NAME parameter 8-3
 - target
 - See target LU
- TERMINATION_COUNT parameter
 - of DISPLAY_MODE verb 10-35
- test for posting
 - using MC_TEST verb 5-79
 - using TEST verb 7-80
- test for REQUEST_TO_SEND notification
 - using MC_TEST verb 5-79
 - using TEST verb 7-80

- TEST parameter
 - of MC_TEST verb 5-79
 - of TEST verb 7-80
- TEST verb 7-80
- TIMEOUT parameter
 - of WAIT_FOR_COMPLETION verb 6-19
- TIMER_EXPIRED return code
- TP_NAME parameter
 - of DEFINE_TP verb 10-21
 - of DELETE verb 10-29
 - of DISPLAY_TP verb 10-39
- TP_NAMES parameter
 - of DISPLAY_LOCAL_LU verb 10-31
- TP_NOT_AVAIL_NO_RETRY
 - See ALLOCATION_ERROR subcodes
- TP_NOT_AVAIL_RETRY
 - See ALLOCATION_ERROR subcodes
- TPN_NOT_RECOGNIZED
 - See ALLOCATION_ERROR subcodes
- TPN parameter
 - of ALLOCATE verb 7-3
 - of MC_ALLOCATE verb 5-3
- transaction
 - definition 1-2
 - distributed processing of 1-2
 - example 1-2
- transaction program
 - application 5-1, 6-1
 - control-operator 7-84
 - defining operating parameters for 10-20
 - definition 1-2
 - displaying operating parameters for 10-39
 - example 1-2
 - execution 1-8
 - instance 1-8
 - LU services component 7-1
 - name
 - carried in allocation request 1-8
 - specified by TPN parameter 5-3, 7-3
 - obtaining attributes of 6-5
 - obtaining properties of 6-5
 - other program statements of 1-8
 - SNA service 7-1, D-1
 - structure 1-8
- truncated data record
 - received by MC_RECEIVE_AND_WAIT verb 5-45
 - received by MC_RECEIVE_IMMEDIATE verb 5-56
- truncated LL field
 - indicated on RECEIVE_AND_WAIT verb 7-47
 - indicated on RECEIVE_IMMEDIATE verb 7-58
- truncation of logical records
 - by SEND_ERROR verb 7-74, 7-75
 - return codes indicating F-11, F-16

- two-way alternate conversation style 2-4, 1-4, 1-9, 2-5, 2-6, 2-11
 - basic conversation verbs 2-11
 - conversation verbs supported for 2-5
 - mapped conversation verbs 2-6
- two-way simultaneous conversation style 2-4, 1-4, 1-9, 2-5, 2-8, 2-13
 - basic conversation verbs 2-13
 - conversation verbs supported for 2-5
 - mapped conversation verbs 2-8
- type-independent conversation verbs 6-1, 2-10, 6-1
 - See also individual verbs
- TYPE parameter
 - of ALLOCATE verb 7-3
 - of DEACTIVATE_CONVERSATION_GROUP verb 9-4
 - of DEACTIVATE_SESSION verb 9-6
 - of DEALLOCATE verb 7-17
 - of GET_TYPE verb 6-7
 - of MC_ALLOCATE verb 5-3
 - of MC_DEALLOCATE verb 5-16
 - of MC_PREPARE_TO_RECEIVE verb 5-37
 - of PREPARE_TO_RECEIVE verb 7-39
 - of SEND_ERROR verb 7-70
- types of symbol strings C-1

U

- UNINTERPRETED_LU_NAME parameter
 - of DEFINE_REMOTE_LU verb 10-16
 - of DISPLAY_REMOTE_LU verb 10-38
- UNKNOWN
 - See ERROR_INDICATION subcodes
- unprotected resources 6-15
- UNRECOGNIZED_MODE_NAME return code
 - for control operator verbs G-4
- UNSUCCESSFUL return code F-5
- user control data
 - received by MC_RECEIVE_AND_WAIT verb 5-45
 - received by MC_RECEIVE_IMMEDIATE verb 5-56
 - sent by MC_SEND_DATA verb 5-65
- USER_CONTROL_DATA_NOT_SUPPORTED return code F-5
- USER_CONTROL_DATA parameter
 - of MC_SEND_DATA verb 5-65
- USER parameter
 - of SIGNOFF verb 11-8

V

- verb
 - ending semicolon 4-2
 - format box 4-2

verb (continued)

name 4-2

parameters

returned 4-3

supplied 4-3

supplied-and-returned 4-3

verb description format 4-1

verbs

basic conversation

See also basic conversation verbs

examples of use B-1

categories of 2-1

control-operator

See control-operator verbs

conversation verbs

See conversation verbs

execution of 1-9

format for describing 4-1

issued by transaction program 1-9

LU services programs use of 7-1

mapped conversation

See mapped conversation verbs

overview description of 2-1

overview descriptions 2-2

remote 2-2

syntax description of 4-1

type-independent conversation

See type-independent conversation verbs

verification

conversation-level security 10-4

list 10-4

resource access 10-22

VOTE_READ_ONLY_PERMITTED parameter

of SET_SYNCPT_OPTIONS verb 6-10

VOTED_READ_ONLY

See OK subcodes

W

WAIT_FOR_COMPLETION verb 6-19

WAIT_FOR_OUTCOME parameter

of SET_SYNCPT_OPTIONS verb 6-10

WAIT_OBJECT_LIST parameter

of WAIT_FOR_COMPLETION verb 6-19

WAIT_OBJECT parameter

of ALLOCATE verb 7-6

of BACKOUT verb 6-2

of CONFIRM verb 7-11

of CONFIRMED verb 7-14

of DEALLOCATE verb 7-18

of FLUSH verb 7-26

of MC_ALLOCATE verb 5-6

of MC_CONFIRM verb 5-10

of MC_CONFIRMED verb 5-13

of MC_DEALLOCATE verb 5-16

WAIT_OBJECT parameter (continued)

of MC_FLUSH verb 5-25

of MC_PREPARE_FOR_SYNCPT verb 5-34

of MC_PREPARE_TO_RECEIVE verb 5-39

of MC_RECEIVE_AND_WAIT verb 5-43

of MC_RECEIVE_EXPEDITED_DATA
verb 5-51

of MC_REQUEST_TO_SEND verb 5-61

of MC_SEND_DATA verb 5-65

of MC_SEND_ERROR verb 5-70

of MC_SEND_EXPEDITED_DATA verb 5-76

of PREPARE_FOR_SYNCPT verb 7-36

of PREPARE_TO_RECEIVE verb 7-40

of RECEIVE_AND_WAIT verb 5-54, 7-45, 7-56

of RECEIVE_EXPEDITED_DATA verb 7-52

of REQUEST_TO_SEND verb 7-62

of SEND_DATA verb 7-66

of SEND_ERROR verb 7-71

of SEND_EXPEDITED_DATA verb 7-77

of SYNCPT verb 6-12

WAIT verb 6-16

waiting for information

using MC_RECEIVE_AND_WAIT verb 5-42

using RECEIVE_AND_WAIT verb 7-44

WHAT_RECEIVED parameter

of MC_RECEIVE_AND_WAIT verb 5-45

of MC_RECEIVE_IMMEDIATE verb 5-56

of RECEIVE_AND_WAIT verb 7-47

of RECEIVE_IMMEDIATE verb 7-57

Communicating Your Comments to IBM

Systems Network Architecture
Transaction Programmer's Reference
Manual for LU Type 6.2

Publication No. GC30-3084-05

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
United States and Canada: **1-800-227-5088**
- If you prefer to send comments electronically, use this network ID:
 - IBM Mail Exchange: **USIB2HPD at IBMMAIL**
 - IBMLink: **CIBMORCF at RALVM13**
 - Internet: **USIB2HPD@VNET.IBM.COM**

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies.

Help us help you!

**Systems Network Architecture
Transaction Programmer's Reference
Manual for LU Type 6.2
Publication No. GC30-3084-05**

We hope you find this publication useful, readable and technically accurate, but only you can tell us! Your comments and suggestions will help us improve our technical publications. Please take a few minutes to let us know what you think by completing this form.

Overall, how satisfied are you with the information in this book?	Satisfied	Dissatisfied
	☐	☐

How satisfied are you that the information in this book is:	Satisfied	Dissatisfied
Accurate	☐	☐
Complete	☐	☐
Easy to find	☐	☐
Easy to understand	☐	☐
Well organized	☐	☐
Applicable to your task	☐	☐

Specific Comments or Problems:

Please tell us how we can improve this book:

Thank you for your response. When you send information to IBM, you grant IBM the right to use or distribute the information without incurring any obligation to you. You of course retain the right to use the information in any way you choose.

Name	Address
------	---------

Company or Organization	
-------------------------	--

Phone No.	
-----------	--

Help us help you!
GC30-3084-05



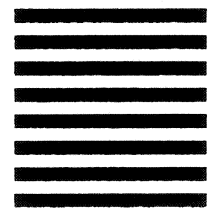
Fold and Tape

Please do not staple

Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
APPC Marketing Enablement III
Department EA6A
PO BOX 12195
RESEARCH TRIANGLE PARK, NORTH CAROLINA 27709-9990



Fold and Tape

Please do not staple

Fold and Tape