DIGITAL SNA APPC/LU6.2 Programming Interface for OpenVMS

Programming

Part Number: AA-ET91G-TE

November 1998

This document describes how to use the DIGITAL SNA APPC/LU6.2 Programming Interface for OpenVMS software to enable an OpenVMS application to exchange messages with a cooperating application on an IBM host.

Revision/Update Information:	This is a revised manual.
Operating System and Version:	OpenVMS VAX Versions 6.2, 7.0, or 7.1 OpenVMS Alpha Versions 6.2, 7.0, or 7.1
Software Version:	DIGITAL SNA APPC/LU6.2 Programming Interface for OpenVMS V2.4

Contents

3 APPC/LU6.2 Programming Interface Features

4 Procedure Calling Format: Conversation Verbs

5 Procedure Calling Format: Control Operator Verbs

Figures

Tables

Preface

The DIGITAL SNA APPC/LU6.2 Programming Interface for OpenVMS enables OpenVMS users to communicate with remote IBM host transaction programs on systems running OpenVMS SNA, or connected to either of the following SNA gateways:

- DECnet SNA Gateway-CT
- DECnet SNA Gateway-ST
- DIGITAL SNA Domain Gateway
- DIGITAL SNA Peer Server
- DIGITAL SNA Access Server for Windows NT

Note

Unless otherwise stated, the term SNA gateway refers to the DECnet SNA Gateway-CT, the DECnet SNA Gateway-ST, the DIGITAL SNA Domain Gateway, the DIGITAL SNA Peer Server, the DIGITAL SNA Access Server for Windows NT, or the OpenVMS SNA (OpenVMS VAX Version 6.2 and Version 7.1 only) when used in this manual.

You can use this interface to develop a complementary transaction program on an OpenVMS system that uses an IBM SNA logical unit (LU) type 6.2 session, to communicate with a corresponding IBM LU6.2 application.

Manual Objectives

The *DIGITAL SNA APPC/LU6.2 Programming Interface for OpenVMS Programming* tells you how to write a transaction on an OpenVMS system to establish an LU-to-LU type 6.2 session with a remote IBM host transaction program.

Intended Audience

This manual is for OpenVMS programmers. To use the DIGITAL SNA APPC/LU6.2 Programming Interface, you need a general understanding of IBM's Systems Network Architecture (SNA) and Advanced Program-to-Program Communication (APPC). For information, see IBM's *An Introduction to Advanced Program-to-Program Communication (APPC)*, Order No. GG24-1584.

Structure of This Manual

This manual has six chapters, nine appendixes, and a glossary.

Chapter 1	DIGITAL SNA application interface products and the features of the APPC/LU6.2 Programming Interface.
Chapter 2	APPC/LU6.2 concepts and terms, and the way the Programming Interface requests services and functions.
Chapter 3	Features of the Programming Interface that help you write and execute your transaction.
Chapter 4	Calling format and parameter list for each conversation verb that the Programming Interface provides.
Chapter 5	Calling format and parameter list for each control operator verb that the Programming Interface provides.
Chapter 6	Procedures for compiling and linking a OpenVMS transaction program using a shareable image.
Appendix A	Summary of the notation that describes parameters in the Programming Interface.
Appendix B	Programming examples in several commonly used languages.
Appendix C	Status codes returned by a conversation verb, and the state change, if any, that a conversation has undergone. Also, status codes that can be returned by a control operator verb.
Appendix D	Table indicating whether a transaction can issue a procedure call when a conversation is in a particular state. Also, state transition information for each APPC/LU6.2 verb.
Appendix E	Table correlating procedures and status messages used by the Programming Interface.
Appendix F	Symbols and values to use when writing your transaction program, if a definition file is unavailable for a particular language.

Appendix G	Features compatible with earlier versions of the Programming Interface.
Appendix H	Status codes that the Programming Interface returns to the OpenVMS transaction program.
Appendix I	Problems you may encounter when using the Programming Interface. Supplements the following: *DECnet SNA Gateway-CT Problem Solving (OpenVMS & ULTRIX)*, *DECnet SNA Gateway-ST Problem Solving (OpenVMS)*, *OpenVMS SNA Problem Solving* and *DIGITAL SNA Peer Server Management*.
Appendix J	Considerations when using the DIGITAL SNA Access Server for Windows NT.
Glossary	Definitions of terms in this manual.

Associated Documents

You may need to refer to the following IBM documents:

- *An Introduction to Advanced Program-to-Program Communication (APPC)*, Order No. GG24-1584. Provides general information about the LU6.2 architecture.
- *CICS/OS/VS* Version 1, Release 7, *Application Programmer's Reference Manual (Command Level)*, Order No. SC33-0241. A general programming reference for CICS Version 1, Release 7; contains information about mapped conversations using CICS.
- *CICS/OS/VS* Version 1, Release 7, *Intercommunication Facilities Guide*, Order No. SC33-0230. Contains information about basic conversations using CICS Version 1, Release 7.
- *Systems Network Architecture Concepts and Products*, Order No. GC30-3072. Provides basic information on SNA for readers wanting an overview of SNA.
- *Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic*, Order No. SC30-3112. Provides comprehensive information on the formats and protocols of SNA nodes.
- *Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*, Order No. SC30-3269. Provides detailed information about formats and protocols for LU6.2.
- *Systems Network Architecture LU6.2 Reference: Peer Protocols*, Order No. SC31-6808.

- *Systems Network Architecture Reference Summary*, Order No. GA27-3136. Provides summary information on SNA formats and sequences.
- *Systems Network Architecture - Sessions Between Logical Units*, Order No. GC20-1868. Contains reference information on SNA formats and protocols for LU types other than 6.2.
- *Systems Network Architecture Technical Overview*, Order No. GC30-3073. Provides a technical description of how SNA functions allow network users to be independent of SNA network characteristics and operations.
- *System Network Architecture Transaction Programmer's Reference Manual for LU Type 6.2*, Order No. GC30-3084. Provides the architectural definition for LU6.2.
- *VTAM Programming for LU6.2*, Version 3, Release 2, Order no. SC30-3400.

You should have the following DIGITAL documents available for reference:

- *DECnet SNA Gateway-CT Installation*
- *DECnet SNA Gateway-CT Management (OpenVMS)*
- *DECnet SNA Gateway-CT Problem Solving (OpenVMS & ULTRIX)*
- *DECnet SNA Gateway-CT Guide to IBM Parameters*
- *DECnet SNA Gateway-ST Installation*
- *DECnet SNA Gateway-CT Management (OpenVMS)*
- *DECnet SNA Gateway-ST Problem Solving (OpenVMS)*
- *DECnet SNA Gateway-ST Guide to IBM Parameters*
- *DIGITAL SNA Domain Gateway Installation*
- *DIGITAL SNA Domain Gateway Management*
- *DIGITAL SNA Domain Gateway Guide to IBM Resource Definition*
- *DIGITAL SNA Peer Server Installation and Configuration*
- *DIGITAL SNA Peer Server Management*
- *DIGITAL SNA Peer Server Network Control Language Reference*
- *DIGITAL SNA Peer Server Guide to IBM Resource Definition*
- *DIGITAL SNA Access Server for Windows NT Installation and Configuration*
- *DIGITAL SNA Access Server for Windows NT Guide to IBM Resource Definition*

- *OpenVMS SNA Installation*
- *OpenVMS SNA Management*
- *OpenVMS SNA Problem Solving*
- *OpenVMS SNA Guide to IBM Parameters*
- *OpenVMS Install Utility Manual*
- *OpenVMS Linker Reference Manual*
- *OpenVMS Linker Utility Manual*
- *OpenVMS Run-Time Library Routines Reference Manual*
- *OpenVMS System Services Reference Manual*
- *Introduction to OpenVMS System Routines*

Graphic Conventions

This manual uses the following conventions:

Convention	Meaning
`Special type`	Indicates an example of either user or system input.
UPPERCASE LETTERS	Represent constant values, or symbols. Code these as specified.
lowercase italics	Represent variables for which you must supply a value.
[ ]	Enclose parameters or symbols that are either optional or conditional. Specify the parameter and value if you want the condition to apply. Do not type the brackets in the line of code. The following rules generally apply to parameters: • You may code or omit an optional parameter. Omitting an optional parameter may impact a related parameter or may cause a default value to be specified. • You may code or omit a conditional parameter. Your choice is determined by how other parameters, in the same or different procedures, are coded.
( )	Enclose a group of values you must specify for a parameter. Type these values in the line of code in the order indicated. Type parentheses exactly where they must appear in a line of code.
Numbers	Decimal unless otherwise noted.
Return	Ends a command line. Unless otherwise specified, end every command line by pressing the Return key.
Ctrl/*x*	Indicates a control character, where *x* is an alphabetic character. Press the Ctrl key and the appropriate alphabetic key simultaneously.

1
Introduction

Using the DIGITAL SNA APPC/LU6.2 Programming Interface or OpenVMS, you can develop complementary OpenVMS transaction programs that exchange messages with remote IBM host transaction programs. To exchange these messages, an OpenVMS transaction program needs support from the Programming Interface when establishing a session with any IBM host using SNA Logical Unit (LU) type 6.2. The Programming Interface uses either one of the SNA gateway products of DIGITAL or the OpenVMS SNA interconnect product for communication between a DIGITAL and an IBM system.

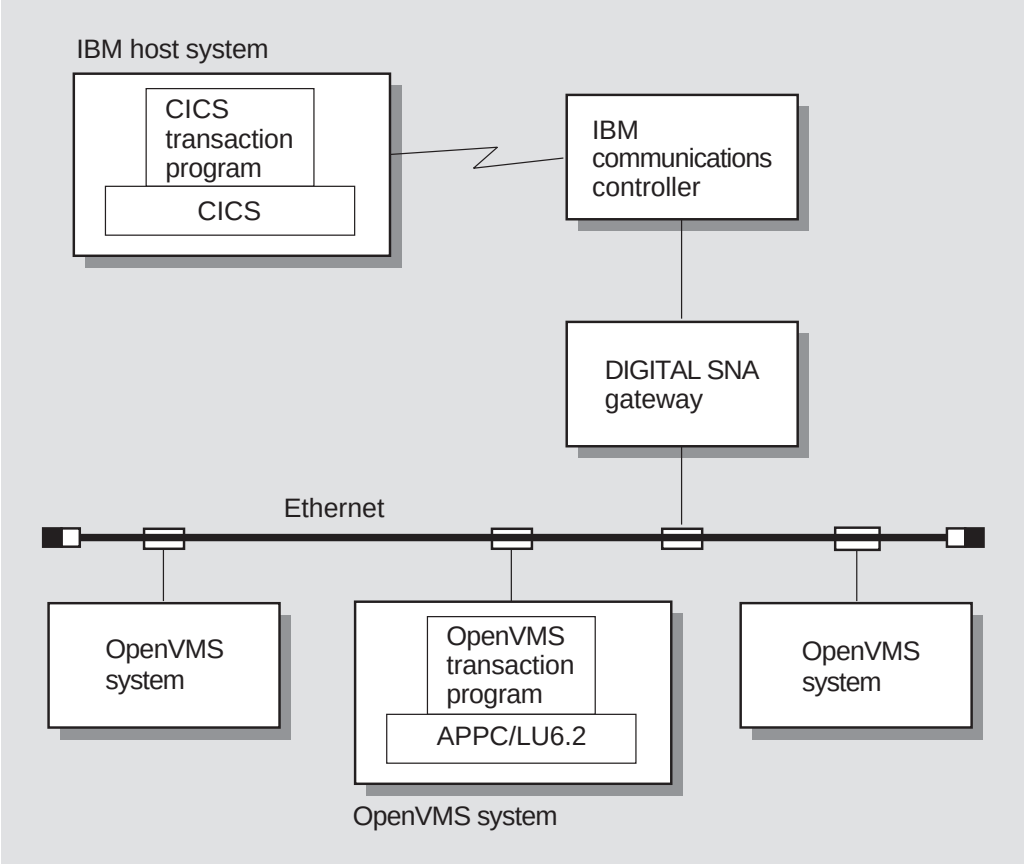
Figure 1–1 shows the connection between an IBM host and a DIGITAL network using the APPC/LU6.2 Programming Interface and a DIGITAL SNA gateway product. The DIGITAL SNA gateway in the figure can be any one of the following gateway products:

- DECnet SNA Gateway-CT
- DECnet SNA Gateway-ST
- DIGITAL SNA Domain Gateway
- DIGITAL SNA Peer Server
- DIGITAL SNA Access Server for Windows NT

DIGITAL SNA Access Server for Windows NT Note

The DIGITAL SNA Access Server for Windows NT product does not fully support the APPC/LU6.2 Programming Interface. In most cases, you will not have to rewrite APPC/LU6.2 programs when using the DIGITAL SNA Access Server for Windows NT. Notes like this one throughout this manual describe any differences when using the DIGITAL SNA Access Server for Windows NT. For a summary of how the DIGITAL SNA Access Server for Windows NT affects the APPC/LU6.2 verbs, see Appendix J.

Figure 1–1 APPC/LU6.2 Using DIGITAL SNA Gateway Connection to IBM Host

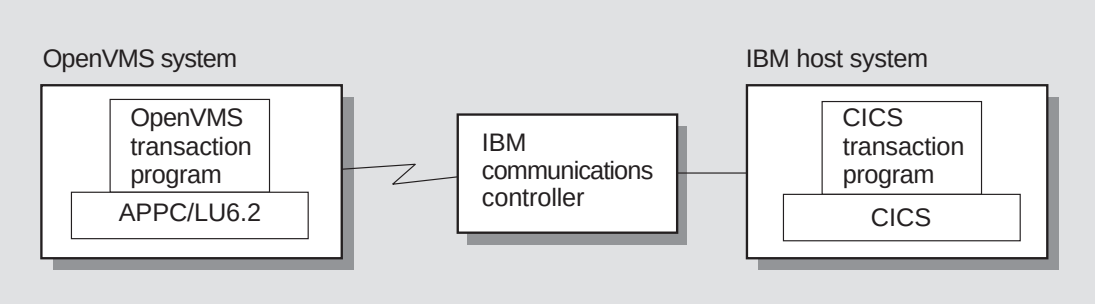


LKG–4418–93R

Figure 1–2 shows a single connection between an OpenVMS system and an IBM SNA host using the OpenVMS SNA product (OpenVMS VAX Version 6.2 and Version 7.1 only) and the APPC/LU6.2 Programming Interface.

The APPC/LU6.2 Programming Interface is one of four DIGITAL SNA for OpenVMS programming interface products. The other three, the DIGITAL SNA 3270 Data Stream Programming Interface, the DIGITAL SNA Application Programming Interface (API) and the Remote 3270 Application Services

Figure 1–2 APPC/LU6.2 Using OpenVMS SNA Connection to IBM Host



LKG–4419–93R

Interface (R3270), enable you to exchange messages with cooperating applications on an IBM host as follows:

- The DIGITAL SNA 3270 Data Stream Programming Interface supports OpenVMS applications in establishing sessions with an LU type 2. This interface can receive uninterpreted 3270 data streams, or it can convert the 3270 data stream into a virtual screen image, which it then updates and returns to the IBM host.
- The DIGITAL SNA Application Programming Interface supports OpenVMS applications that need to establish sessions with LU types 0, 1, 2, and 3.
- The Remote 3270 Application Services Interface (R3270) supports OpenVMS applications that generate and interpret all instances of the 3270 data stream for sessions with LU types 0, 1, 2, and 3.

To use the APPC/LU6.2 Programming Interface, your IBM host system must support SNA LU type 6.2 sessions and either CICS/ESA or VTAM Version 3.2 or later. Refer to the Software Product Description (SPD) for details.

If your system does not support LU type 6.2 sessions, one of the other DIGITAL SNA programming interfaces may be able to meet your programming needs.

For more information about IBM's advanced program-to-program communication, see *An Introduction to Advanced Program-to-Program Communication (APPC)*, Order No. GG24-1584.

1.1 Logical Unit Type 6.2

Logical Unit 6.2 is a general purpose architecture that enables IBM products to communicate with one another. Unlike other IBM LUs, which are designed with specific products in mind, LU6.2 has general function as the goal, that is, a common LU for all products.

The LU6.2 architecture defines a set of protocols. To communicate with one another, products must implement LU6.2 according to these protocols. They can, however, implement that definition in different ways. There are two general implementations of LU6.2:

- "Open-box" implementations
- "Closed-box" implementations

"Open-box" implementations allow users to write their own communications code to solve business problems. The DIGITAL SNA APPC/LU6.2 Programming Interface for OpenVMS and IBM's CICS/ESA, Intersystem Communication (ISC) are examples of open-box implementations.

For open-box implementations, IBM's architectural definition for LU6.2 provides a set of procedures, also called verbs—that programmers use to design distributed transactions. CICS ISC implements the verb functions with EXEC CICS commands. OpenVMS transaction programs, using the APPC/LU6.2 Programming Interface, implement the verb functions with OpenVMS procedures. Whether products use verbs, EXEC CICS commands, or procedures, they implement the IBM architecture and enable cooperating transactions to communicate. Programmers use the various procedures to make calls from their transaction program. The procedures function as subroutines within the transaction; that is, they execute their function and then return control to the transaction program.

LU6.2 provides the procedures for transaction programs to call. Transaction programs written in different programming languages can use the LU6.2 procedures to exchange messages. Regardless of the language they are written in, the transaction programs must cooperate. Then, as long as both sides of the communication use the LU6.2 interface and can handle the data, the transactions can communicate with each other.

"Closed-box" implementations apply to turnkey products. The user cannot develop customized solutions for business problems. The MR/SNADS Facility and IBM's DISOSS/370 are examples of closed-box implementations.

1.2 APPC/LU6.2 Programming Interface Features

Using the APPC/LU6.2 Programming Interface, an OpenVMS transaction program can:

- Perform peer-to-peer communications (distributed transaction processing) as opposed to traditional hierarchical host-to-device communications
- Perform a wide range of distributed transaction processing, provided the cooperating transaction uses identical record formats and functions

Communication using LU6.2 is analogous in function to the DECnet task-to-task communication over a DECnet logical link. As with DECnet, communication between transaction programs using LU6.2 is transparent. You can move to a remote IBM host transaction program without knowing the details of the underlying networks.

LU6.2 transaction programs exchange information by means of a conversation, a temporary logical path established between two cooperating transaction programs. As in DECnet task-to-task communication, transaction programs must cooperate during the transmission process. For instance, when one transaction program issues a command to send data, the other transaction program must issue a command to receive the data. In cooperating transaction programs, you can make calls to the APPC/LU6.2 Programming Interface procedures to:

- Establish LU 6.2 sessions
- Set up conversations between transaction programs
- Send and receive data
- Support mapped conversations
- Support basic conversations
- Confirm data exchange

For information about tearing down conversations and terminating LU6.2 sessions on OpenVMS systems, see the *Introduction to OpenVMS System Routines*.

Note

The DIGITAL SNA APPC/LU6.2 Programming Interface for OpenVMS does not support the following functions:

- Conversations with a synchronization level of SYNCPT
- Backout and Sync Level Syncpoint procedures

- Parallel sessions (The Microsoft SNA Server does support parallel sessions. Because all session control occurs in the Microsoft SNA Server software, the DIGITAL SNA APPC/LU6.2 Programming Interface for OpenVMS product effectively supports parallel sessions when using a Microsoft SNA Server.)
- Control operator procedures other than DEFINE_REMOTE, DEFINE_TP, ACTIVATE_SESSION, DEACTIVATE_SESSION, DELETE_REMOTE and DELETE_TP

You can find the calling format for each of the procedures that the Programming Interface supports in Chapter 4 and Chapter 5.

1.3 SNA Concepts

You can develop a variety of transaction programs using the Programming Interface. Because the interface sends, receives, and interprets SNA protocol messages on behalf of the OpenVMS transaction program, you need not be concerned with SNA message formats and protocols. Nevertheless, this manual assumes a general knowledge of SNA and an awareness of LU type 6.2 architecture. For further details, see *An Introduction to Advanced Program-to-Program Communication (APPC)*, Order No. GG24-1584, and the *System Network Architecture Programmer's Reference Manual for LU Type 6.2*, Order No. GC30-3084.

1.4 Common Interface Transaction Programs

You can use the Programming Interface to write transaction programs that make up the secondary logical unit (SLU) half-session partner in an LU type 6.2 session. For example, you can write transactions that do the following:

- Communicate with IBM's Distributed Office Support System (DISOSS) on the IBM host.
- Write cooperating LU-to-LU transaction programs that communicate directly with each other. Traditionally, to communicate with an IBM mainframe using a non-IBM product, end users have had to emulate specific IBM devices with an application. With the advent of LU6.2, peer-to-peer communications are possible.

DIGITAL SNA Access Server for Windows NT Note

When using the Microsoft SNA Server, you can write transaction programs that make up the primary logical unit (PLU) half-session partner.

2

Concepts and Terms

The DIGITAL SNA APPC/LU6.2 Programming Interface for OpenVMS, (called the APPC/LU6.2 Programming Interface), enables an OpenVMS transaction program and a remote IBM host transaction program to take part in a distributed transaction environment that uses IBM's Advanced Program-to-Program Communication.

A transaction involves at least two participants: a requester of a processing operation and an executor of the operation. The request can be accompanied by input data, and successful completion of the operation may include the return of output data. In a distributed transaction, the program that requests the operation and the program that performs the operation can be on different computer systems and at different facilities and sites. The programs communicate through the facilities of a computer network.

In a DIGITAL SNA distributed transaction environment, programs communicate by means of the combined facilities of TCP/IP or DECnet, SNA, and an SNA gateway. In a pure SNA configuration, each program is assigned a logical unit (LU) that performs communication functions and provides access to the network. The APPC/LU6.2 Programming Interface provides LU6.2 functions for a transaction program running on an OpenVMS system.

This chapter discusses how to:

- Establish sessions and conversations
- Send and receive messages
- Synchronize partner programs
- Deallocate (terminate) a conversation

2.1 Sessions and Conversations

Before your transaction program and a remote transaction program can exchange messages, they must first establish a conversation. A conversation is a short-term logical connection that lasts only for the duration of one complete transaction. (A transaction can be of any duration, depending upon the amount of data the program has to transmit.) Neither your transaction program nor the remote transaction program controls the conversation. Peer-to-peer communication, rather than hierarchical communication, characterizes these transactions.

Conversations take place over sessions between logical units (LUs). A session is a long-term logical connection that permits communication between two logical units. When your transaction program establishes a conversation, the interface allocates a session for your transaction program to use for the duration of your transaction program's conversation with the remote transaction program. When your transaction program completes, the conversation ends. The session, however, remains active and available for other conversations.

2.1.1 Requesting a Session for a Conversation

To request a session for a conversation, the OpenVMS transaction program calls the SNALU62$ALLOCATE procedure. By default, the APPC/LU6.2 Programming Interface attempts to allocate an active session for the conversation.

- If a session is available (that is, if one exists and is not currently allocated to another conversation), the APPC/LU6.2 Programming Interface starts the conversation immediately.
- If a session exists but is not currently available, or if an active session does not currently exist, the APPC/LU6.2 Programming Interface automatically activates one and allocates it for the conversation.

Both the OpenVMS transaction program and the IBM transaction program can allocate a conversation; thus, both inbound and outbound conversation allocation is possible. In order for an IBM transaction program to initiate a conversation with an OpenVMS transaction program, the OpenVMS transaction program must activate a session between the two LUs using the SNALU62$ACTIVATE_SESSION verb.

2.1.2 Basic and Mapped Conversations

IBM defines two types of conversation, basic and mapped, for advanced program-to-program communication. The APPC/LU6.2 Programming Interface supports both types with the same set of procedures described in this manual. In the APPC/LU6.2 Programming Interface procedures that permit a choice of conversation types, choose the type you want by specifying the appropriate symbolic code—either SNALU62$K_BASIC_CONVERSATION or SNALU62$K_MAPPED_CONVERSATION—in the *type* parameter of the SNALU62$ALLOCATE verb.

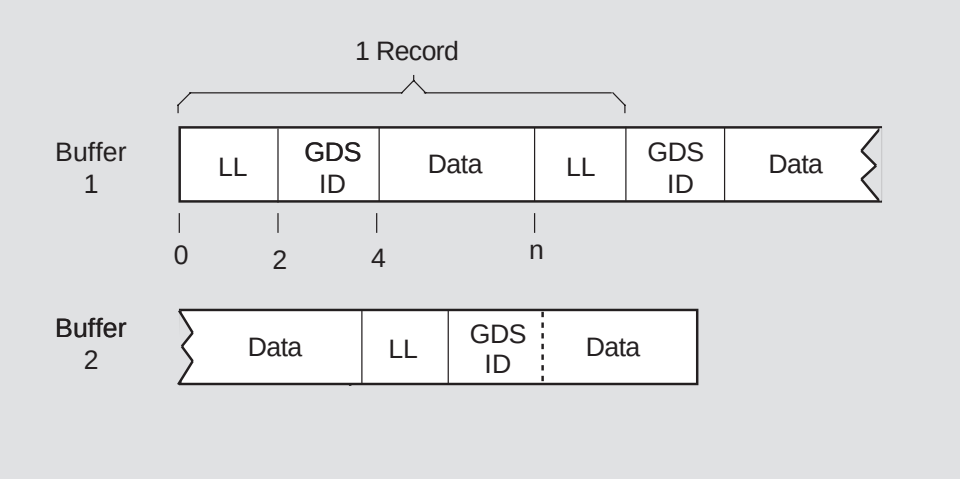
2.1.2.1 Generalized Data Stream (GDS)

Both basic and mapped conversations use the generalized data stream (GDS) to transmit data. The GDS is a standard data stream in LU6.2 communications. Because mapped conversations manage the GDS for you, DIGITAL recommends that you use them. For more information about the GDS, see *An Introduction to Advanced Program-to-Program Communication (APPC)*, Order No. GC24-1584, *IBM Transaction Programmer's Reference Manual for LU Type 6.2*, Order No. GC30-3084, and the *IBM CICS/ESA, Application Programmer's Reference Manual (Command Level)*, Order No. SC33-0241.

2.1.2.2 Basic Conversations

Basic conversations are suitable for system programmers who need to manipulate data-stream information that is exchanged. Transaction programs using basic conversations are responsible for performing error recovery and data-stream mapping, and for building and interpreting the GDS headers when used. When the OpenVMS transaction program uses GDS headers, it supplies a buffer containing one or more logical records. Each record is preceded by a 2-byte header indicating the length (LL) of the record you are transmitting (including the 2-byte header), and the data. The LL is a 16-bit integer in IBM format; that is, the first byte is the most significant byte and the second byte is the least significant byte. In the first two bytes of the data, you have the option of supplying GDS identifiers. These identifiers provide information about the format and meaning of the following data. Figure 2–1 illustrates the basic conversation buffer. In the illustration, please note that the buffer can contain more than one record and that GDS ID may or may not appear.

Figure 2–1 Basic Conversation Buffer



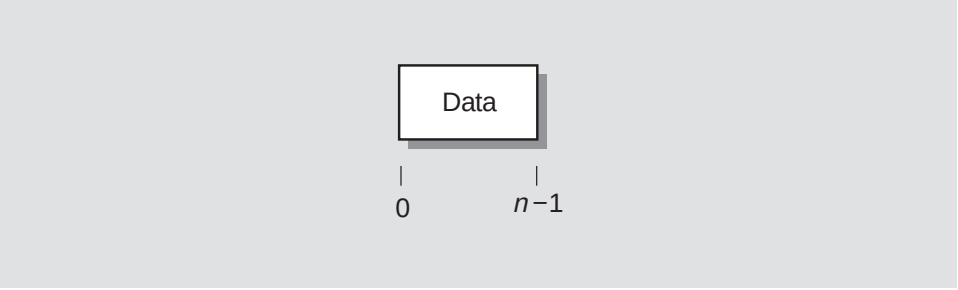
LKG–4562–93R

2.1.2.3 Mapped Conversations

Mapped conversations ease the work of writing transaction programs because they provide a higher level interface than basic conversations. They provide additional functions for the user-written transaction program such as data-stream handling, detection of remote program errors, and processing of local program errors. When you use mapped conversations, you supply a buffer of data; the APPC/LU6.2 Programming Interface transforms all data being sent to another transaction program into the GDS, and IBM LU services restores the data to its original form before the destination transaction program receives it. With mapped conversations, the OpenVMS transaction program is not aware of the GDS headers and is not responsible for using or interpreting them. Figure 2–2 illustrates the mapped conversation buffer.

For more information about conversations, see *An Introduction to Advanced Program-to-Program Communication (APPC)*, Order No. GG24-1584.

Figure 2–2 Mapped Conversation Buffer



LKG–4563–93R

2.2 Sending Messages

Sending a message to the remote IBM host transaction program involves three separate actions:

- Putting the OpenVMS transaction program into the send state
- Submitting the data to an APPC/LU6.2 Programming Interface buffer
- Transmitting the contents of the APPC/LU6.2 Programming Interface buffer to the remote IBM host transaction program

2.2.1 Putting the OpenVMS Transaction Program into the Send State

LU6.2 uses states to synchronize conversations (that is, to ensure that the transaction programs call procedures in the proper sequence) between two cooperating transaction programs. A state reflects the conditions of a conversation at any given moment. For example, for communication to take place, one transaction must be in send state and the other in receive state. The state of a conversation determines what procedures the OpenVMS transaction program can call. If the OpenVMS transaction program attempts to call a procedure when it is not in the proper state (that is, when it is not synchronized with the cooperating transaction), the error the procedure will return is SNALU62$_STAERR. When both transaction programs are in the correct state, the conversations are synchronized. Successfully calling a procedure changes the state of the conversation.

To send a message to the remote IBM transaction program, the OpenVMS transaction program must be in the send state and the remote IBM host transaction program must be in the receive state. If the OpenVMS transaction program wishes to send a message but is currently in the receive state, it must call the SNALU62$RECEIVE_AND_WAIT procedure and wait for the cooperating transaction to send a SEND notification. The SEND notification will be received by way of the SNALU62$RECEIVE_AND_WAIT procedure.

The OpenVMS transaction program can also issue the procedure SNALU62$REQUEST_TO_SEND and the following steps will occur:

1. The OpenVMS transaction program calls the SNALU62$REQUEST_TO_SEND procedure. The APPC/LU6.2 Programming Interface transmits a request-to-send to the remote IBM host transaction program.
2. The remote IBM host transaction program receives the request-to-send and at some point returns a SEND notification to the OpenVMS transaction program, granting the request. Note that the IBM Host transaction program can also either ignore or defer granting the request.
3. The OpenVMS transaction program calls the SNALU62$RECEIVE_AND_WAIT procedure and tests the *what_received* parameter until it receives the SNALU62$K_SEND symbol.

   Note

   The *what_received* parameter may contain other symbols that place the conversation in the same or in a different state and require different action (see Section 4.1.1).

4. The OpenVMS transaction program is now in the send state and the remote IBM host transaction program is in the receive state.

For this sequence to succeed, the remote IBM host transaction program must respond exactly as shown here. This response is not automatic: the remote IBM host transaction program can respond in a manner different from that described.

Upon successful completion of the SNALU62$ALLOCATE procedure (described in Section 2.1.1), the OpenVMS transaction program is in the send state and can transmit data to the remote IBM host transaction program. Once the OpenVMS transaction program enters the receive state, it cannot return to the send state until after the SEND notification is delivered by means of a subsequent SNALU62$RECEIVE_AND_WAIT procedure.

2.2.2 Submitting Data for Transmission

In the course of a conversation, the OpenVMS transaction program may wish to submit record data to the APPC/LU6.2 Programming Interface for transmission to the remote IBM host transaction program. The way data is submitted depends on whether the transactions are engaged in a mapped conversation or a basic conversation.

An OpenVMS transaction program engaged in a mapped conversation submits records to the APPC/LU6.2 Programming Interface one at a time. To submit a single data record, the OpenVMS transaction program calls the SNALU62$SEND_DATA procedure and provides a buffer containing the data record. This record consists entirely of data. The data can include an SNA function management header (FMH). An FMH is an optional header included in SNA request units to provide information about the presentation of data at the destination transaction program.

An OpenVMS transaction program engaged in a basic conversation can submit records to the APPC/LU6.2 Programming Interface one at a time or in groups. To submit one or more records, the OpenVMS transaction program calls the SNALU62$SEND_DATA procedure and provides a buffer containing one or more logical records. Each logical record consists of a field (LL) 2 bytes long, followed by a data field.

For both types of submission, the APPC/LU6.2 Programming Interface transfers the data record(s) from the OpenVMS transaction program's buffer to a buffer of its own. When the buffer is filled, the APPC/LU6.2 Programming Interface sends the contents to the remote IBM host transaction program in a single transmission.

2.2.3 Transmitting the Send Buffer

By default, the APPC/LU6.2 Programming Interface buffers the data records, commands, and control information that the OpenVMS transaction program wants to send to the remote IBM host transaction program. When the buffer is full, the APPC/LU6.2 Programming Interface sends the contents to the remote IBM host transaction program in a single transmission.

In some cases, the OpenVMS transaction program may wish to submit a record or message for immediate transmission. To do this, the transaction program can call the SNALU62$FLUSH procedure. The APPC/LU6.2 Programming Interface immediately transmits the contents of the buffer to the remote IBM host transaction program; however, the OpenVMS transaction program does not leave the send state.

The procedures SNALU62$CONFIRM, SNALU62$RECEIVE_AND_WAIT, SNALU62$SEND_ERROR, and SNALU62$PREPARE_TO_RECEIVE can also make the APPC/LU6.2 Programming Interface flush its send buffer. These procedures are described in the sections that follow.

2.3 Receiving a Request-to-Send Notification

At any point during a conversation, the remote IBM host transaction program can send a request-to-send to the APPC/LU6.2 Programming Interface indicating that it wants to send a message to the OpenVMS transaction program. This message can be data, conversation status information, or a confirmation request.

To notify the OpenVMS transaction program that the remote IBM host transaction program has sent a request-to-send, the APPC/LU6.2 Programming Interface uses the *rts_rec* parameter provided by the following procedures: SNALU62$CONFIRM, SNALU62$RECEIVE_AND_WAIT, SNALU62$SEND_DATA, and SNALU62$SEND_ERROR.

The complete operation consists of the following steps:

1. The APPC/LU6.2 Programming Interface receives a request-to-send from the remote IBM host transaction program. It holds the request, and at some point, does the following:
   - Calls one of the following procedures:
     - SNALU62$CONFIRM
     - SNALU62$RECEIVE_AND_WAIT
     - SNALU62$SEND_DATA
     - SNALU62$SEND_ERROR
   - Specifies a location in the parameter list to receive a TRUE/FALSE *rts_rec* indicator resulting from the procedure called
2. When the APPC/LU6.2 Programming Interface returns a value of TRUE to the *rts_rec* parameter, the OpenVMS transaction program now has a choice. It can:
   - Issue a receive, and agree to the request, or
   - Issue a send, and ignore the request.

2.4 Receiving a Message from the IBM Host Transaction Program

To receive a message from the remote IBM host transaction program, the OpenVMS transaction program calls the SNALU62$RECEIVE_AND_WAIT procedure. If a message is available, APPC/LU6.2 passes it to the calling transaction at once. If not, the transaction waits until a message arrives from the remote IBM host transaction program.

The OpenVMS transaction program can issue an SNALU62$RECEIVE_AND_WAIT request when it is in either the receive or send state. If it is currently in the send state, the APPC/LU6.2 Programming Interface immediately flushes the send buffer and transmits the contents to the remote IBM host transaction program, along with a send indicator to put the remote host transaction program into the send state.

The SNALU62$RECEIVE_AND_WAIT procedure provides a way for an OpenVMS transaction program to enter the receive state and receive a message from the remote IBM host transaction program. APPC/LU6.2 provides four other procedures that the OpenVMS transaction program can use to perform similar operations. These are the SNALU62$RECEIVE_IMMEDIATE procedure, the SNALU62$PREPARE_TO_RECEIVE procedure, the SNALU62$POST_ON_RECEIPT procedure, and the SNALU62$WAIT procedure.

1. The SNALU62$RECEIVE_IMMEDIATE procedure provides the simplest way for an OpenVMS transaction program to enter the receive state. If a message is available, APPC/LU6.2 passes it at once to the calling transaction. The OpenVMS transaction program does not have to wait for a message from the remote IBM host transaction program before it can enter the receive state.
2. The SNALU62$PREPARE_TO_RECEIVE procedure causes the APPC/LU6.2 Programming Interface to flush the send buffer and send a SEND notification to the remote IBM host transaction program. The remote transaction enters the send state when it receives the SEND notification. The APPC/LU6.2 Programming Interface always flushes the send buffer when the OpenVMS transaction program issues the SNALU62$PREPARE_TO_RECEIVE procedure.
3. The SNALU62$POST_ON_RECEIPT procedure allows you to watch for received data on multiple conversations. SNALU62$POST_ON_RECEIPT causes the Programming Interface to notify the specified conversation when information is available for receipt by the OpenVMS transaction

program. The information can be data, conversation status, or a request for confirmation. Notification can occur in three ways:

- Setting the event flag parameter (if available)
- Calling a user-specified AST routine
- Allowing a call to a pending SNALU62$WAIT procedure to complete.

4. The SNALU62$WAIT procedure suspends execution of the OpenVMS transaction program and waits for notification to occur on any conversation from a list of conversations. The user must issue the SNALU62$RECEIVE_AND_WAIT procedure to transfer the data into the buffer.

Note

If the conversation is in the send state, the OpenVMS transaction program must issue the SNALU62$PREPARE_TO_RECEIVE procedure before issuing the SNALU62$POST_ON_RECEIPT procedure.

2.5 Confirmation Processing

In an LU6.2 conversation, a confirmation is an exchange of messages that enables two transaction programs to agree that a particular operation either has succeeded or failed. If the operation has failed, the confirmation includes the exchange of error messages indicating the reason for the failure.

Using the APPC/LU6.2 Programming Interface, an OpenVMS transaction program can request a confirmation from the remote IBM host transaction program and can respond positively to a confirmation request sent by the remote IBM host.

To request a confirmation, the OpenVMS transaction program calls the SNALU62$CONFIRM procedure, specifies a conversation, and waits for a response. When the procedure completes, the status vector contains the response.

To receive a confirmation request from the remote IBM host transaction program, the OpenVMS transaction program uses the SNALU62$RECEIVE_AND_WAIT procedure (see discussion of the *what_received* parameter in Section 4.11.

To send a positive response to a confirmation request, the OpenVMS transaction
program calls the SNALU62$CONFIRMED procedure. The APPC/LU6.2 Programming Interface sends the confirmation message to the remote IBM host transaction program.

2.6 Sending and Receiving Error Notification

At some point in a conversation, the OpenVMS transaction program or the remote IBM host transaction program may need to inform its partner that it has detected an error.

To report an error, the OpenVMS transaction program calls the SNALU62$SEND_ERROR procedure. The parameter list includes a type indicator that the OpenVMS transaction program uses to specify whether it is reporting an end-user transaction program error or an LU services error (a software error that occurs with basic conversations).

If the remote IBM host transaction program reports an error, the APPC/LU6.2 Programming Interface returns the message to the next procedure called by the OpenVMS transaction program. Errors are returned in a status vector specified by the OpenVMS transaction program (see Section 3.1.3).

2.7 Ending a Conversation

To end a conversation, the OpenVMS transaction program calls the SNALU62$DEALLOCATE procedure. When the procedure completes, the conversation ends. The session, however, remains active and available for other conversations.

You can deallocate a conversation in three ways, although not all ways are legitimate in every conversation state:

- Normal deallocation initiated by the OpenVMS transaction program
- Abnormal deallocation initiated by the OpenVMS transaction program
- Local deallocation initiated by the OpenVMS transaction program after the remote IBM host transaction program has deallocated the conversation

2.7.1 Normal Deallocation

The OpenVMS transaction program initiates a normal deallocation. To perform a normal deallocation, the OpenVMS transaction program must be in the send state after either completing a receive with the *what_received* parameter equal to SNALU62$K_SEND or having completed a send request. There are two types of normal deallocation; flush and sync-level:

- In a flush deallocation (*type* equals SNALU62$K_FLUSH), the Programming Interface first sends all data in its internal buffer and then deallocates the conversation.
- In a sync-level deallocation (*type* equals SNALU62$K_SYNC_LEVEL), the APPC/LU6.2 Programming Interface sends all data in its internal buffer but requests confirmation if the sync-level is CONFIRM. If the confirm response is positive, the conversation is deallocated. If the confirm response is negative, the SNALU62$DEALLOCATE procedure completes with an error code such as program error purging (SNALU62$_PRERPU) or program error truncating (SNALU62$_PRERTR). The remote IBM host transaction program returns an error code indicating what went wrong. The OpenVMS transaction program is now in receive state and the remote IBM host transaction program is now in send state. The way the OpenVMS transaction program recovers from the error is program dependent.

2.7.2 Abnormal Deallocation

The OpenVMS transaction program initiates abnormal (ABEND) deallocation. To perform an abnormal deallocation, the APPC/LU6.2 Programming Interface can be in send, receive, or confirm state. Abnormal deallocation is a method of reporting program failure to the remote IBM host transaction program. There are three types of abnormal deallocation, and DIGITAL supports two: abend program and abend SVC.

- Abend program (*type* equals SNALU62$K_ABEND_PROG) indicates that the OpenVMS transaction program has detected a fatal error, such as insufficient memory or no output device available, and that the abend program cannot continue.
- Abend SVC (*type* equals SNALU62$K_ABEND_SVC) indicates that the protocol agreed to by the two cooperating transaction programs was violated and that the abend program does not know how to continue.

Note

Abend program or abend SVC are application specific. On the IBM side of the communication, the IBM host transaction program issues abend program and the LU issues abend SVC. The APPC/LU6.2 Programming

Interface does not have this restriction, so the OpenVMS programmer can use the two types of abnormal deallocation to convey the kind of program failure that has occurred.

DIGITAL SNA Access Server for Windows NT Note

On mapped conversations using the DIGITAL SNA Access Server for Windows NT, the abend program and abend SVC options are changed to the generic AP_ABEND option supported by the Microsoft SNA Server.

2.7.3 Local Deallocation

The OpenVMS transaction program initiates local deallocation. When the remote IBM host transaction program deallocates its end of a conversation with a normal or abend deallocation, the OpenVMS transaction program enters deallocate state. As described in Section 2.7.1, there are two types of normal deallocations, and the response of the OpenVMS transaction program depends on which type the remote IBM host transaction program uses.

- For the flush deallocation, the conversation actually enters the deallocate state when the OpenVMS transaction program is notified by means of the status returned on a receive.
- For the sync-level deallocation, the conversation enters the deallocate state when the OpenVMS transaction program responds positively to the confirmation request.

The abend deallocation is reported by means of the status returned on a receive, as in normal (flush) deallocation.

Once in the deallocate state, the OpenVMS transaction program can issue the deallocate local (*type* equals SNALU62$K_LOCAL). If a conversation is in deallocate state and you do not issue a deallocate local, the SNA session you were using can still be used for other conversations. The local deallocation frees virtual memory. If the OpenVMS transaction program does not issue a local deallocation, memory used by APPC/LU6.2 is not freed, but failing to issue a local deallocation does not tie up the SNA session.

3

APPC/LU6.2 Programming Interface Features

The DIGITAL SNA APPC/LU6.2 Programming Interface for OpenVMS provides features that assist you in writing and executing your OpenVMS transaction program. The Programming Interface performs the SNA communications function, while you concentrate on your application program(s). APPC/LU6.2 features include the following:

- Mechanisms for returning status information
- Support for DECnet and TCP/IP gateway nodes and OpenVMS/SNA
- Program initialization parameters
- Synchronous and asynchronous operation
- Session activation
- Security information
- Supplying access information to the IBM host
- Outbound conventions
- Contention polarity
- Session failure notification
- Conversation deallocation notification
- LU security support

3.1 Returning Status Information

The APPC/LU6.2 Programming Interface uses two mechanisms to return status codes to the OpenVMS transaction program:

- Function value returns
- An I/O status vector

For a description of all the status codes returned by the Programming Interface, see Appendix H.

3.1.1 Function Value Returns

When an APPC/LU6.2 Programming Interface procedure finishes its attempt to perform an operation, it returns a function value to indicate whether the operation succeeded or failed. It places this value in register R0. After each call to an APPC/LU6.2 Programming Interface procedure, you must check this status value.

Each high-level language provides some mechanism for testing the return status value in R0. Often you need only check the low-order bit, such as by a test for TRUE (success or information return) or FALSE (error or warning return).To check the entire value for a specific return condition, each language provides a way for your program to determine the values associated with specific symbolically defined codes. Always use these symbolic names when you write tests for specific conditions.

3.1.2 The I/O Status Vector

All APPC/LU6.2 Programming Interface procedures return completion status to the OpenVMS transaction program by means of a status vector that provides the transaction with complete information about error conditions. The status vector contains a top-level completion code that is the same as the function value and, in certain cases, supplies additional qualifying codes.

The format of the status vector (see Figure 3–1) is identical to that of the $PUTMSG message vector used by OpenVMS.

3.1.3 Using Status Vectors

If an error occurs, each component of the network involved can pass a message to the APPC/LU6.2 Programming Interface, which uses this information to build the status vector.

Usually, the transaction program displays the error by means of the OpenVMS system service call to $PUTMSG. $PUTMSG translates the status vector into a human-readable message and sends it to a terminal or file.

$PUTMSG uses the following format:

SYS$PUTMSG *msgvec [,actrtn][,facnam][,actprm]*

where

msgvec	is the message argument vector. It contains the address of the status vector.
actrtn	is a user-supplied action routine executed during message processing.
facnam	is a facility prefix used in the first or only message written by $PUTMSG.
actprm	is a parameter passed to the action routine.

For further information about dealing with errors, see the sections "Condition Handling" and "$PUTMSG" of the *OpenVMS System Services Reference Manual*.

If you do not want to call $PUTMSG, you can use LIB$SIGNAL or LIB$STOP, by means of a call to LIB$CALLG, to generate a signal indicating that an exception condition has occurred in your program. If your high-level language does not allow you to do this, you should use the $PUTMSG system service to display the messages.

LIB$CALLG uses the following format:

LIB$CALLG *argument list, procedure*

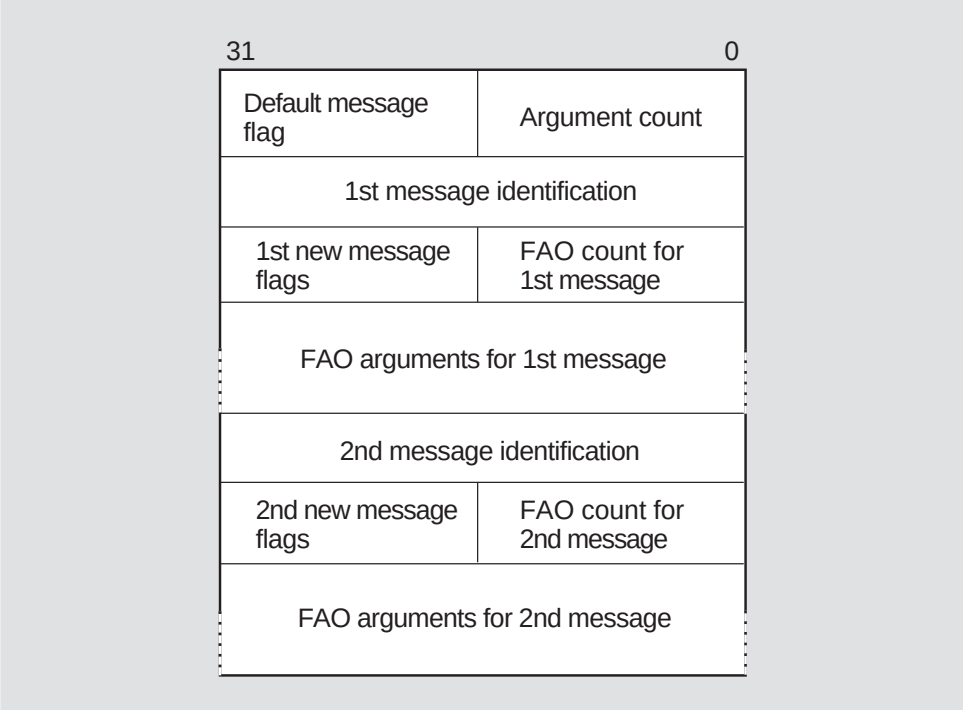
where

argument list	is the status vector
procedure	is LIB$SIGNAL or LIB$STOP

For further information about LIB$CALLG, LIB$SIGNAL, and LIB$STOP, see the *OpenVMS Run-Time Library Routines Reference Manual*.

The APPC/LU6.2 Programming Interface returns status vectors in the format shown in Figure 3–1.

Figure 3–1 I/O Status Vector



LKG–8087–93R

The following list provides a description of the fields that make up the status vector:

- Argument count

  Specifies the total number of longwords in the status vector, excluding the longword that contains the argument count and default message flags.

- Default message flags

  Specifies a mask defining the portions of the message(s) to be requested. If a mask is not specified, the process default message flags are used. If a mask is specified, it is passed to $GETMSG as the FLAGS argument. For further information, see "Get Message" in the *OpenVMS System Services Reference Manual.*

  This mask establishes the default flags for each message in the call until a new set of flags (if any) is specified. Each specified "new message flags" field sets a new default.

Bits 4 through 15 must be zeros.

- Message identification

  32-bit numeric value that uniquely identifies a message. Messages can be identified by symbolic names defined for system return status codes, OpenVMS RMS status codes, and so on.

- FAO count

  Number of Formatted ASCII Output ($FAO) arguments, if any, that follow a message in the status vector. For further information, see $FAO in the *OpenVMS System Services Reference Manual*.

- New message flags

  New mask for the $GETMSG flags that defines a new default for the message and all subsequent messages.

- FAO arguments

  FAO arguments required by the message.

3.2 Specifying the SNA Gateway

The SNALU62$DEFINE_REMOTE procedure uses the parameter *gwynode* to determine to which SNA gateway the APPC interface should connect. Depending on your gateway and on your network environment, you can use this parameter to specify one of the following:

- a name that is dynamic and can be interpreted as a DECnet node name (all gateways except the DIGITAL SNA Access Server for Windows NT), an internet node name (all gateways except the DIGITAL SNA Access Server for Windows NT), or an LU6.2 Server name (DIGITAL SNA Access Server for Windows NT only). See Section 3.2.1.
- a DECnet node name (all gateways except the DIGITAL SNA Access Server for Windows NT. See Section 3.2.2.
- an internet node name (all gateways except the DIGITAL SNA Access Server for Windows NT. See Section 3.2.3.
- an LU6.2 Server name (DIGITAL SNA Access Server for Windows NT only). See Section 3.2.4.

3.2.1 Specifying a Dynamic Gateway Name

By default, the APPC interface attempts to dynamically interpret the supplied gateway name. It interprets the gateway name using the following process:

1. Checks for an LU6.2 Server name

   When you specify a dynamic gateway name, the interface first checks to see if you are supplying a LU6.2 Server name directly. To do this, the interface scans the supplied gateway node name for the presence of a slash (/) character. If the slash character is found, the interface interprets the supplied gateway node name as an LU6.2 Server name as described in Section 3.2.4.

2. Checks for a reference to a OpenVMS logical specifying the gateway node name information

   If the slash character is not found, the interface checks to see if you are referencing a OpenVMS logical that indirectly specifies the gateway node name information. The interface uses the supplied node name to construct the OpenVMS logical SNALU62$*node-name*_GWY. The interface then tests for the existence of this logical. If this logical is defined, the interface translates the logical and scans the value of the logical for the presence of a slash (/) character. If the slash character is found, the interface interprets the supplied gateway node name as an LU6.2 Server name as described in Section 3.2.4.

3. Assumes a DECnet or internet node name and checks for transport order logical

   If the checks for the existence of an LU6.2 Server name specification fail (either directly or during the scan of the SNALU62$*node-name*_GWY logical), the interface assumes that the node name must be a DECnet or internet node name.

   Note

   By now, the interface knows it is not dealing with a DIGITAL SNA Access Server for Windows NT because the gateway name is not an LU6.2 Server name specification (there is no slash character).

   The interface then checks for the existence of the SNA_TRANSPORT_ ORDER logical. This value of this logical directs the interface which transports to use to reach the gateway and in what order to try the transports. The following values are possible:

`decnet,tcp`	Attempt a DECnet connection first. If this fails, attempt a TCP/IP connection.
`tcp,decnet`	Attempt a TCP/IP connection first. If this fails, attempt a DECnet connection.
`decnet,notcp`	Attempt only a DECnet connection.
`nodecnet,tcp`	Attempt only a TCP/IP connection.

If the SNA_TRANSPORT_ORDER logical is undefined, the interface tries the DECnet transport first, then the TCP/IP transport.

3.2.2 Specifying a DECnet Node Name

DIGITAL SNA Access Server for Windows NT Note

This option is not available for the DIGITAL SNA Access Server for Windows NT.

To force the APPC interface to interpret the supplied gateway node name as a DECnet node name (and therefore use exclusively DECnet transport protocols), specify a valid DECnet node name followed by a double colon. For example, to force the APPC interface to use DECnet transport protocols to reach the DECnet node BOSTON, specify the gateway node name `BOSTON::`.

Alternatively, you can leave off the double colon and specify the value of the SNA_TRANSPORT_ORDER logical as `decnet,notcp`.

Note that you can specify a DECnet node name either directly or indirectly (through the use of the SNALU62$*node-name*_GWY logical).

3.2.3 Specifying a Internet Node Name

DIGITAL SNA Access Server for Windows NT Note

This option is not available for the DIGITAL SNA Access Server for Windows NT.

To force the APPC interface to interpret the supplied gateway node name as a internet node name (and therefore use exclusively TCP/IP transport protocols), specify a valid internet node name followed by a single colon. For example, to force the APPC interface to use TCP/IP transport protocols to reach the internet node `boston.cmp.com`, specify the gateway node name `boston.cmp.com:`.

Alternatively, you can leave off the single colon and specify the value of the SNA_TRANSPORT_ORDER logical as `nodecnet,tcp`.

Note that you can specify a internet node name either directly or indirectly (through the use of the SNALU62$*node-name*_GWY logical).

3.2.4 Specifying an LU6.2 Server Name

Note

This option is only available for the DIGITAL SNA Access Server for Windows NT.

To force the APPC interface to interpret the supplied gateway node name as an LU6.2 Server name, specify the gateway node name using the following format:

node-name/[*server-name*/*transport*]

where:

node-name	Specifies the DECnet or internet node name where the LU6.2 Server is located.
server-name	Specifies the name of the LU6.2 Server. Note that this is the LU6.2 Server name, not the server's network service name. If this field is not specified, the interface uses the name LU62_SRV.
transport	Specifies the transport to use to reach the LU6.2 Server. If this field is not specified, the interface uses DECnet transports.

Note that to specify exclusively an LU6.2 Server name you must include at least one slash character in the supplied gateway node name.

3.2.5 Specifying the Gateway's TCP/IP Port Information

If you have either explicitly or implicitly selected the TCP/IP transport, you must define the TCP/IP port on the gateway to which the interface should connect. The method you use to do this depends on the gateway you are using.

3.2.5.1 For a DIGITAL SNA Access Server for Windows NT

If you are using a DIGITAL SNA Access Server for Windows NT system, you must define the LU6.2 Server's network service name locally in the DIGITAL TCP/IP Services for OpenVMS database. On the DIGITAL SNA Access Server for Windows NT, an LU6.2 Server Name is defined using the "Manage Services" dialog box. When an LU6.2 Server Name is defined, the Access Server automatically creates a network service name using the LU6.2 Server Name with a suffix of _CLI. The port number is assigned dynamically by the DIGITAL SNA Access Server for Windows NT software and displayed as you create the service. If you fail to note the port number during Access Server configuration, the network service name and TCP/IP port number are stored in

the file %SystemRoot%\system32\drivers\etc\SERVICES on the Windows NT system.

On an OpenVMS system, you add a network service name by using the SET SERVICE command in the UCX utility provided with the DIGITAL TCP/IP Service for OpenVMS software. For example, to define the service LU62_SRV_CLI, use the following command:

SET SERVICE LU62_SRV_CLI /PORT=8001 /PROCESS=LU62_SRV /PROTOCOL=TCP /USER=NONE /FILE=NL

The /PORT value must match the port number on the Access Server. The /PROCESS qualifier must be set to a unique value (for example, the LU6.2 Server Name). The /USER and /FILE parameters are required but can be set to any value. To improve troubleshooting, you should use the Access Server's network service name as the service name on the OpenVMS system.

For more information about how to change the DIGITAL TCP/IP Services for OpenVMS database, see the DIGITAL TCP/IP Services for OpenVMS product documentation.

3.2.5.2 For a DIGITAL Gateway Other Than the DIGITAL SNA Access Server for Windows NT

If you are using a DIGITAL gateway other than a DIGITAL SNA Access Server for Windows NT system, you can use the SNA_TCP_PORT logical to specify the LU6.2 TCP/IP port number on the gateway. For example, if you want the interface to connect to TCP/IP port number 1234 on the gateway, define the SNA_TCP_PORT logical as follows (you must use the number sign (#) as shown:

```
$ define SNA_TCP_PORT #1234
```

If you have defined a TCP/IP service name in the DIGITAL TCP/IP Services for OpenVMS database that references the desired TCP/IP port number, you can use this service name as the logical value. For example, if you have the service name LU62GWY defined in the database, you can define the SNA_TCP_PORT logical as follows:

```
$ define SNA_TCP_PORT LU62GWY
```

The default connection TCP/IP port number is 108.

3.3 Program Initialization Parameters

Program Initialization Parameters (PIPs) are a means of passing initialization parameters or environment setup information to the remote IBM transaction program. For example, your application may require that the output of the remote transaction program be directed to a printer or screen display. The content of a PIP variable is application dependent. The procedure SNALU62$GET_PIP is used to obtain PIP variables that were supplied by the remote transaction.

3.3.1 Sending PIP Data to a Remote Transaction Program

PIP parameters are supplied to a remote transaction program when an OpenVMS transaction program allocates a conversation. The SNALU62$SUPPLY_PIP procedure allocates an internal data structure to hold PIP information; it stores the address of the data structure in the *pip_context* variable that is supplied on a call to SNALU62$ALLOCATE. PIP parameters are sent to the remote transaction program when the sending conversation first goes into receive state.

3.3.2 Receiving PIP Data from Remote Transaction Program

The OpenVMS transaction program receives notification that the remote transaction program has initiated a conversation to it. The SNALU62$GET_PIP procedure returns the PIP parameters that were specified by the remote transaction program.

3.4 Synchronous and Asynchronous Operation

Different transaction programs require different amounts of cooperation between the sender and receiver of information. To satisfy these differences, a transaction program that calls an APPC/LU6.2 Programming Interface procedure can specify one of two modes of operation: synchronous and asynchronous.

There are two categories of verbs:

1. Verbs that do not contain the EFN or ASTADR parameters

   These verbs are totally synchronous in operation. The user calls the verb, and then when the verb returns, the operation is complete. The verb will never cause the program to go into a wait state. An obvious exception is SNALU62$WAIT, which is meant to wait and, therefore, can only be specified in synchronous mode.

The synchronous-only verbs are as follows:
SNALU62$CONFIRMED
SNALU62$DEFINE_REMOTE
SNALU62$DEFINE_TP
SNALU62$FLUSH
SNALU62$GET_ATTRIBUTES
SNALU62$GET_PIP
SNALU62$GET_TYPE
SNALU62$POST_ON_RECEIPT
SNALU62$RECEIVE_IMMEDIATE
SNALU62$SUPPLY_PIP
SNALU62$WAIT

2. Verbs that contain the EFN and ASTADR parameters

   These verbs can be used either synchronously or asynchronously:

   - **Synchronous mode** — The verb functions in synchronous mode if both the EFN and ASTADR parameters are omitted (or 0 is supplied) on the call to the verb.
   - **Asynchronous mode** — The verb functions in asynchronous mode if the EFN and/or the ASTADR parameter(s) are supplied on the call to the verb.

   The verbs in this category are:
   SNALU62$ACTIVATE_SESSION
   SNALU62$DEACTIVATE_SESSION
   SNALU62$ALLOCATE
   SNALU62$CONFIRM
   SNALU62$DEALLOCATE
   SNALU62$DELETE
   SNALU62$PREPARE_TO_RECEIVE
   SNALU62$RECEIVE_AND_WAIT
   SNALU62$SEND_DATA
   SNALU62$SEND_ERROR

3.4.1 Synchronous Mode

In synchronous or wait mode, the following steps occur:

1. The OpenVMS transaction program calls a procedure and provides the required list of parameters. If the parameters are invalid, step 2 occurs. If the parameters are valid, step 3 occurs.

2. The APPC/LU6.2 Programming Interface returns status information immediately as a function value, with additional information in the status vector. The OpenVMS transaction program can resume execution.
3. The APPC/LU6.2 Programming Interface sends the request to the IBM host transaction program, and suspends the OpenVMS transaction program.
4. The remote cooperating transaction program sends information back to the OpenVMS transaction program.
5. The APPC/LU6.2 Programming Interface procedure returns a function value to indicate the success or failure of the operation. The procedure also places the status code and other information in the status vector. The OpenVMS transaction program can resume execution.

3.4.2 Asynchronous Mode

In asynchronous mode, the transaction program issues a call to request an operation and immediately resumes execution. It does not wait for the operation to be completed. For this reason, transaction programs that call procedures asynchronously must specify an event flag (see Section 3.4.2.1 or provide a completion procedure that the APPC/LU6.2 Programming Interface can call to indicate that it has completed its attempt to perform the operation).

The completion procedure is an asynchronous system trap (AST). For additional information about the AST, event flag services, and AST services, see the *OpenVMS System Services Reference Manual*. An asynchronous call involves the following steps:

1. The OpenVMS transaction program calls a procedure and requests an operation.
2. The procedure immediately returns a status code as a function value. If success is returned, step 4 occurs. If failure is returned, step 3 occurs.
3. The procedure places a status code and other information in the status vector. The APPC/LU6.2 Programming Interface does not attempt to perform the operation. The transaction program resumes execution with the completion of the operation outstanding.
4. At the completion of the operation, the APPC/LU6.2 Programming Interface performs the following:
   - Fills in the status vector with completion information indicating success or failure.
   - Sets an event flag.

- Calls a completion procedure to inform the transaction program that the APPC/LU6.2 Programming Interface has finished its attempt to perform the requested operation.

Notes

- APPC/LU6.2 uses AST routines in its internal layers to make data available for presenting to the user application. These AST routines are temporarily blocked when a user application AST routine, such as a notify, attach, or verb completion routine is executing. Therefore, be careful when using APPC/LU6.2 verbs in an AST routine.

  For example, a SNALU62$RECEIVE_IMMEDIATE call in an attach routine that returns "NDAVAIL" (no data is available), will continue to return that status while called from the attach routine. This will prevent data from being made available due to the occupied AST level. (In this situation, use a SNALU62$POST_ON_RECEIPT with specified AST completion routine in the attach routine and then exit the attach routine.)
- An AST cannot be interrupted by another AST. ASTs are queued and serviced sequentially. Therefore, the notify routine cannot be interrupted by completion ASTs and completion ASTs cannot be interrupted by the notify routine.
- In asynchronous mode, no other conversation verb can be issued while an asynchronous verb is in progress for a specific session.
- Only verbs in category 2 (Section 3.3) can be both synchronous and asynchronous. Verbs specified without the *efn* and *astadr* asynchronous parameters cannot be called as synchronous verbs within ASTs.
- If an asynchronous verb is called using the synchronous format (that is, without the *efn* and *astadr* parameters specified) and ASTs are disabled, then within the call, ASTs will be temporarily reenabled.

Both synchronous and asynchronous procedures have the following format:

status=SNALU62$*procedure_name* (*parameters*)

where:

status	is a status code entered in general register R0 when a procedure returns. It indicates successful completion or error condition.
SNALU62$procedure_name	is the name of the APPC/LU6.2 Programming Interface asynchronous procedure you want to call.
parameters	is a list of information needed to perform the requested procedure.

Both conversation and operator control verbs can be used asynchronously. However, for asynchronous completion, the parameter list must include *efn* and/or *astadr*:

efn	An event flag set when notification of asynchronous completion becomes due.
astadr	The address of a caller's AST routine when notification of asynchronous completion becomes due.
astprm	A parameter to pass to the AST routine (by reference) when notification occurs. You can use the *astprm* parameter to provide a pointer to a data structure containing the resource ID in multithreaded applications.

3.4.2.1 Event Flags

Event flags should be allocated before use. Use the LIB$GET_EF (or LIB$RESERVE_EF) and LIB$FREE_EF Run Time Library routines to handle allocation and deallocation.

With the limited number of event flags, it is possible to use event flags that have not been specifically allocated. Due to this it is possible to have multiple asynchronous operations sharing these limited resources. Problems may occur when one asynchronous operation sets a shared event flag, as this may indicate to other users of that event flag that their operation has completed.

If there are not enough event flags for the number of conversations, use the $SYNC routine and specify the event flag and cleared (unique) status_ vector (as iosb). Use a different event flag/status_vector combination for each conversation. $SYNC will not complete until both the event flag is set and the status_vector is filled in. See the *OpenVMS System Services Reference Manual*, and the *OpenVMS Run-Time Library Routines Reference Manual* for more information.

3.5 Session Activation

The APPC/LU6.2 Programming Interface has two ways of activating a session:

1. **Active connect** — Initiate a session by requesting the remote LU to send a BIND. This brings a session up immediately. Request session activation through the SNALU62$DEFINE_REMOTE procedure *initiate_type* parameter, with the value SNALU62$K_INITIATE_ONLY specified. A subsequent call to either SNALU62$ACTIVATE_SESSION or SNALU62$ALLOCATE activates the session.

   Note

   The SNALU62$ALLOCATE verb will also allocate a conversation.

2. **Passive connect** — Listen on a specific LU for a BIND to arrive. The session does not come up immediately. Wait for session activation using the SNALU62$DEFINE_REMOTE procedure, *initiate_type* parameter, with the value SNALU62$K_INITIATE_OR_QUEUE specified. A subsequent call to either SNALU62$ACTIVATE_SESSION or SNALU62$ALLOCATE waits for the IBM TP to activate the session.

   Note

   The SNALU62$ALLOCATE verb will also allocate a conversation.

DIGITAL SNA Access Server for Windows NT Note

When a session activation request with a DIGITAL SNA Access Server for Windows NT fails, your program may not receive the exact error returned by the host IBM system. You must use the Windows NT event log viewer to determine the exact cause of the session activation failure.

3.6 Session Deactivation

The APPC/LU6.2 Programming Interface has several ways to deactivate a session using either of two verbs.

The SNALU62$DEACTIVATE_SESSION verb deactivates a specific session. The verb supports both normal and cleanup deactivation. If the argument type is SNALU62$K_DEACT_NORMAL, the session deactivates after conversation ends. The synchronous form of SNALU62$DEACTIVATE_SESSION, called with argument type SNALU62$K_DEACT_NORMAL, should only be called if there is no current conversation for the session.

If the argument type is SNALU62$K_DEACT_CLEANUP, the session deactivates immediately regardless of the current state of the conversation or session. Any outstanding verb is canceled, and its AST routine is called.

Note

If the outstanding verb is SNALU62$POST_ON_RECEIPT, its AST routine is not called because this is a notification AST and not a completion AST.

A second verb, SNALU62$DELETE, deactivates all sessions associated with the specified LU name. SNALU62$DELETE returns only after the current conversation has ended.

3.7 Security

You can use session-level security from your OpenVMS transaction program, but only if the IBM host system is running CICS Version 1, Release 7, or later.

- Session-level security has a mechanism to password-protect an LU on the IBM system. Only OpenVMS applications that know the password can establish an SNA session with that LU.
- Inbound conversation-level security has a mechanism to allow certain users to use specified transaction programs on the IBM system. The IBM system manager sets up a transaction program such that only a particular set of users can access that transaction program. When the OpenVMS transaction program allocates a conversation to the IBM transaction program, it must supply a valid USERNAME/PASSWORD pair.

The APPC/LU6.2 Programming Interface has two functions to assist you with security: partner-LU verification and partner-end-user verification.

- Partner-LU verification is the session level security provided through the SNALU62$ACTIVATE_SESSION and SNALU62$DEFINE_REMOTE procedures. Password and logon information can be specified in parameters to the SNALU62$DEFINE_REMOTE procedure. See Chapter 5 for additional information on these procedures.
- Partner-end-user verification is inbound conversation-level security that takes place at the time a conversation starts. You can specify access security information in parameters to the SNALU62$ALLOCATE procedure. The remote LU uses this access security information to validate access to the program and resources. See Chapter 4 for additional information on this procedure.

DIGITAL SNA Access Server for Windows NT Note

The DIGITAL SNA Access Server for Windows NT does not itself support partner-LU verification. This level of verification is provided by the Microsoft SNA Server.

3.8 Defining IBM Access Information

To establish a session with a remote IBM host transaction program, the OpenVMS transaction program must supply the IBM host with the following access information:

- **Physical Unit (PU) identification**. A value identifying the gateway PU, or the OpenVMS SNA PU (for example, SNA-0) used to establish the session. For OpenVMS SNA, DECnet SNA Gateway-CT, and DECnet SNA Gateway-ST only.
- **Session address**. A value indicating the SLU that you want to use to establish a session with the IBM host. For OpenVMS SNA, DIGITAL SNA Domain Gateway-ST, and DIGITAL SNA Domain Gateway-CT only.
- **Logical Unit (LU) identification**. A value identifying the gateway LU used to establish the session. For DIGITAL SNA Domain Gateway-ST, DIGITAL SNA Domain Gateway-CT, and DIGITAL SNA Peer Server only.
- **Application name**. A character string identifying the PLU application (for example, CICS) that you want to connect to in the IBM host.

- **Logon mode name**. A string specifying an entry in a logon mode table that gives a set of BIND parameters for the session.
- **LU-LU password**. A string used for session level LU-LU verification during session activation.

For complete details about IBM access information for the SNA gateway and OpenVMS SNA, see *OpenVMS SNA Management*, *DIGITAL SNA Domain Gateway Management*, *DIGITAL SNA Peer Server Management*, *DIGITAL SNA Access Server for Windows NT Installation and Configuration*, or *OpenVMS SNA Installation*.

Providing access information involves two steps:

1. An OpenVMS transaction program calls the SNALU62$DEFINE_REMOTE procedure, specifies a local name for the IBM LU, and supplies a list of access information as parameters. The APPC/LU6.2 Programming Interface associates the locally defined name with the access information.
2. An OpenVMS transaction program calls the SNALU62$ALLOCATE or SNALU62$ACTIVATE_SESSION procedure and includes the locally defined LU name in the parameter list. The APPC/LU6.2 Programming Interface uses the access information associated with the name to establish a session to support one or more transaction conversations.

3.9 Contention Polarity

Contention for session resources occurs when both partners attempt to begin a conversation simultaneously on the same session. The contention is resolved according to the polarity agreed upon by both partners when the session was established. The contention winner (first speaker) always receives the session resources and the contention loser (bidder) always has to wait.

The DIGITAL SNA APPC/LU6.2 Programming Interface, allows the OpenVMS transaction program to be either the contention winner or the contention loser.

If the OpenVMS transaction is the contention loser, a call to SNALU62$ALLOCATE will not complete until the contention has been resolved. If the IBM transaction is also attempting to begin a conversation, the allocate request will complete with a resource failure retry error.

3.10 Outbound Conversations

Outbound conversations enable an IBM transaction program to initiate a conversation. In order for the remote transaction program to initiate a conversation, an SNA session must:

- Be already activated by the OpenVMS program
- Be waiting for an attach request from a remote IBM transaction program to a transaction program that has been defined by a call to the SNALU62$DEFINE_TP procedure.

The OpenVMS program must use the SNALU62$DEFINE_TP verb to define the name of the transaction program with which the IBM program initiates a conversation. When the IBM transaction program initiates a conversation, the attach routine specified on the SNALU62$DEFINE_TP verb is called. If the remote IBM TP attempts to initiate a conversation with a transaction program that is undefined by the OpenVMS transaction program, then conversation initiation fails. The attach routine is called with the *resource_id* parameter; the new conversation is then in receive state. The *resource_id* parameter passed to the routine also affects subsequent conversation verbs.

3.11 Notification of Session Failure

The APPC/LU6.2 Programming Interface has the ability to notify the OpenVMS transaction of session failure. If a verb is outstanding at the time of the session failure, status is returned on that verb indicating the nature of the failure. If no verb is outstanding, the OpenVMS transaction is notified through the notify routine and the status is returned through the notify block. The notification parameters can be specified on the SNALU62$ACTIVATE_SESSION verb. See Chapter 5 for more details on the session notification parameters.

3.12 Notification of Conversation Deallocation

The APPC/LU6.2 Programming Interface has the ability to notify the OpenVMS transaction of the receipt of a conversation deallocation request. If a verb is outstanding at the time of the deallocation, status is returned on that verb. If no verb is outstanding, the the notification routine is called with the notification parameter. Notification parameters can be specified on the SNALU62$ALLOCATE verb.

DIGITAL SNA Access Server for Windows NT Note

The DIGITAL SNA Access Server for Windows NT does not support conversation deallocation notification. Any notification parameters specified on the SNALU62$ALLOCATE verb are ignored.

The notification parameters available on the SNALU62$ACTIVATE_SESSION and SNALU62$ALLOCATE verbs provide two separate functions, notification of session termination and conversation termination, respectively. To utilize both of these functions, call SNALU62$ACTIVATE_SESSION before calling SNALU62$ALLOCATE.

Note

Since the conversation notification can notify an application that a DEALLOCATE has been received while the application is receiving other data, Digital Equipment Corporation recommends using the notification only to note that a DEALLOCATE has been received. The conversation is not actually deallocated until a call to either SNALU62$RECEIVE_AND_WAIT or SNALU62$RECEIVE_IMMEDIATE returns with one of the DEALLOCATE status values. Failure to continue to process receives will leave the conversation active (locally) and may prevent further processing for that session.

3.13 Gateway LU Security Support

The interface, along with the SNA gateway, also supports LU security. You can use the *authorization_password* argument for SNALU62$DEFINE_REMOTE to specify the password assigned to a particular LU or set of LUs on the SNA gateway. In addition to *authorization_password*, you can also protect gateway LUs by assigning a node name, user id, and terminal id. These parameters are obtained internally by the interface and passed to the gateway. The user ID is obtained from the current process unless an alternate process ID is specified. See the *DIGITAL SNA Domain Gateway Management*, or the *DIGITAL SNA Peer Server Management* manuals, for additional information on LU security.

4

Procedure Calling Format: Conversation Verbs

This chapter describes the calling format, status codes, and transition states for conversation procedures provided by the Programming Interface. The following procedures are available:

- SNALU62$ALLOCATE
- SNALU62$CONFIRM
- SNALU62$CONFIRMED
- SNALU62$DEALLOCATE
- SNALU62$FLUSH
- SNALU62$GET_ATTRIBUTES
- SNALU62$GET_PIP
- SNALU62$GET_TYPE
- SNALU62$POST_ON_RECEIPT
- SNALU62$PREPARE_TO_RECEIVE
- SNALU62$RECEIVE_AND_WAIT
- SNALU62$RECEIVE_IMMEDIATE
- SNALU62$REQUEST_TO_SEND

- SNALU62$SEND_DATA
- SNALU62$SEND_ERROR
- SNALU62$SUPPLY_PIP
- SNALU62$WAIT

Calls to these procedures have the following general format:

status=SNALU62$procedure name ([argument],...,argument)

where

status	is a status code returned as a function value.
procedure_name	is the procedure that you want to call.
( )	delimits the argument list.
argument	is a symbol containing information that the OpenVMS transaction program passes to the Programming Interface. Shorthand notation describes the argument's characteristics. Appendix A summarizes the notation.
[*argument*]	indicates an optional argument.

Arguments pass to the Programming Interface in two ways:

- By **reference** (or address). The argument is the address of an area or field that contains the value. An argument passed by address is usually expressed as a reference name or label associated with an area or field.
- By **descriptor**. This argument is also an address for a special data structure called a descriptor (see the *Introduction to OpenVMS System Routines*).

In this chapter, the argument definitions for each procedure specify how each argument is passed.

Which procedures an OpenVMS transaction program calls depends upon the state of the conversation. When an OpenVMS transaction program calls a procedure, the state of a conversation may change as a result of

- The function of the procedure
- The result of a procedure called by the remote IBM host transaction program
- Network errors

The following conversation states are possible when you call one of the procedures described in this chapter:

- **Reset** — the OpenVMS transaction program can allocate a conversation.

- **Send** — the OpenVMS transaction program can send data or request confirmation.
- **Receive** — the OpenVMS transaction program can receive information from the remote IBM host transaction program.
- **Confirm** — the OpenVMS transaction program can reply to a request for confirmation.
- **Deallocate** — the remote IBM host transaction program has deallocated the conversation and the OpenVMS transaction program can deallocate the conversation locally.

Figure D–1 shows the conversation state transitions that can occur when a program issues a conversation verb. The figure correlates each verb to a conversation state.

4.1 SNALU62$ALLOCATE

The SNALU62$ALLOCATE procedure initiates a basic or mapped conversation between a local OpenVMS transaction program and a remote IBM host transaction program. The conversation requires an SNA session to support it. By default, the SNALU62$ALLOCATE procedure attempts to allocate an active available LU6.2 session for the conversation. If such a session does not exist, the procedure establishes an LU6.2 session and then allocates it to the conversation.

Format:

status.wlc.v=SNALU62$ALLOCATE (*resource_id.wlu.r*,
status_blk.wx.dx,
[*lu_name.rlu.r*],
other.rt.dx,
[*mode_name.rt.dx*],
tpn.rt.dx,
[*type.rlu.r*],
[*ret_ctrl.rlu.r*],
[*sync_level.rlu.r*],
[*security.rlu.r*],
[*user.rt.dx*],
[*password.rt.dx*],
[*profile.rt.dx*],
[*pip_context.rlu.r*],
[*efn.rlu.r*],
[*astadr.szem.r*],
[*astprm.rlu.r*],
[*notify_rtn.szem.r*],
[*notify_prm.rlu.r*],
[*polarity.rlu.r*])

Arguments:

status
When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits together represent the severity of the error. Returned by function value.

resource_id
A location to receive an ID value assigned to the conversation by the APPC/LU6.2 Programming Interface. This ID must be supplied on all subsequent verb calls for this conversation. Passed by reference.

status_blk
A longword status vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.

lu_name
A value indicating whether the partner transaction program is located at the local LU or at a remote LU. This parameter takes the following value:

- SNALU62$K_OTHER indicates that the partner program is located at another LU (the default).

Passed by reference.

other
A locally defined name for the remote LU. Valid only if SNALU62$K_OTHER is specified for the *lu_name* parameter. This value must be a locally defined name using the SNALU62$DEFINE_REMOTE verb. The name is associated with a remote *lu_name* as well as the information necessary to place the conversation on a session. Passed by descriptor as an ASCII string.

mode_name
A logon mode table entry name to be passed to the remote LU. The *mode_name* parameter designates the network properties for the session to be allocated for the conversation. If *mode_name* is not specified, the APPC/LU6.2 Programming Interface uses the one specified in the SNALU62$DEFINE_REMOTE procedure. The *mode_name* parameter overrides a name specified by means of the SNALU62$DEFINE_REMOTE procedure. Passed by descriptor as an eight-character ASCII string.

DIGITAL SNA Access Server for Windows NT Note: The value for the *mode_name* parameter must match a mode name defined on the Microsoft SNA Server.

tpn
An identifier of the remote IBM host transaction program that the OpenVMS transaction program wants to engage in an LU6.2 conversation. This identifier is not translated. Passed by descriptor as an EBCDIC string.

type

This value specifies the type of conversation to be allocated:

- SNALU62$K_BASIC_CONVERSATION specifies a basic conversation.
- SNALU62$K_MAPPED_CONVERSATION specifies a mapped conversation (the default).

For information on basic and mapped conversations, see Chapter 2, "Concepts and Terms." Passed by reference.

ret_ctrl

A value indicating when the the APPC/LU6.2 Programming Interface performs the SNALU62$ALLOCATE operation and returns control to the OpenVMS transaction program. The parameter takes the following value:

- SNALU62$K_WHEN_SESSION_ALLOC Allocate a session to the conversation and then return control to the OpenVMS transaction program. This is the default.

Passed by reference.

sync_level

A value specifying the synchronization level that the local OpenVMS transaction program and the remote IBM host transaction program can use on this conversation. This parameter takes the following values:

- SNALU62$K_SL_NONE This value indicates that the program will not perform confirmation processing or sync-point processing on this conversation.
- SNALU62$K_SL_CONFIRM This value indicates that the program can perform confirmation processing but not sync-point processing on this conversation. SNALU62$K_SL_ CONFIRM is the default.

Passed by reference.

security

A value specifying access security information that the remote LU uses to validate access to the remote program and its resources. This parameter is valid only if you are using CICS, Version 1.7 or later. The parameter can take the following values:

- SNALU62$K_NONE specifies that access security information is not specified in this allocation request.
- SNALU62$K_SAME specifies that you are to use access security information defined by a previous conversation on the same session.

  If no access security information has been previously defined, the Programming Interface will use the user ID of the current process, or the process specified by the PID parameter on SNALU62$DEFINE_REMOTE. The length of the user ID must not exceed 10 characters or security will be downgraded to "none." If an acceptable user ID has been specified or is obtained from the OpenVMS process, it is passed to the host as an "already verified" user ID.
- SNALU62$K_PGM specifies that you are to use access security information supplied in the USER_ID, PASSWORD, and PROFILE parameters to this verb. Specify a value for each parameter as a string descriptor, 10 alphanumeric characters or less in length.

Passed by reference.

user_id

A value that identifies the end-user making the allocation request. Valid only if SNALU62$K_PGM is specified on the *security* parameter. Passed by descriptor.

password

The remote LU uses this value and the user ID to verify the identity of the end-user making the allocation request. Valid only if SNALU62$K_PGM is specified on the *security* parameter. Passed by descriptor.

profile

The remote LU uses this value, in addition to or in place of the user ID, to determine which remote resources the local program can access, and which resources the remote program can access. Valid only if SNALU62$K_PGM is specified on the *security* parameter. Passed by descriptor.

DIGITAL SNA Access Server for Windows NT Note: The *profile* parameter is ignored by the Microsoft SNA Server.

pip_context

pip_context is a context variable that allows the OpenVMS transaction program to supply PIP parameters to the remote transaction program. The actual PIP parameters are specified by a call to the SNALU62$SUPPLY_PIP procedure. *pip context* is returned by SNALU62$SUPPLY_PIP. Passed by reference.

efn	An event flag to set when notification becomes due. If this parameter is omitted, event flag 0 is set. Passed by reference.
astadr	The address of an AST routine to call when notification becomes due. Passed by reference.
astprm	A parameter to pass to the AST routine when notification occurs. This AST routine is called with this parameter only. Passed by reference.
notify_rtn	Notify service routine to be executed when the APPC interface receives a DEALLOCATE message from the remote application and cannot report either on the current verb or its absence. Passed by reference. **DIGITAL SNA Access Server for Windows NT Note:** The *notify_prm* parameter is ignored by the Microsoft SNA Server.
notify_prm	Notify parameter to be passed to the notify service routine specified by the *notify_rtn* parameter. Passed by reference. The user-written notify routine has the following calling format: notify_rtn(notify_prm.rlu.r) **DIGITAL SNA Access Server for Windows NT Note:** The *notify_prm* parameter is ignored by the Microsoft SNA Server.
polarity	Specifies the contention polarity for the session. SNALU62$K_FIRST_SPEAKER indicates the local LU will be the contention winner. SNALU62$K_BIDDER indicates that the local LU will be the contention loser. SNALU62$K_FIRST_SPEAKER is the default value. Passed by reference.

4.1.1 Status Codes

The SNALU62$ALLOCATE procedure returns the following successful completion code:

- SNALU62$_OK

The SNALU62$ALLOCATE procedure returns the following error codes:

- SNALU62$_ALLERR
- SNALU62$_DEALNOR
- SNALU62$_PARERR
- SNALU62$_RESFRET
- SNALU62$_STAERR

- SNALU62$_UNSUC

DIGITAL SNA Access Server for Windows NT Note

If no session is available, the SNALU62$ALLOCATE procedure performs the function of the SNALU62$ACTIVATE_SESSION procedure. In this case, if the error occurred before the DIGITAL SNA Access Server for Windows NT attempted to connect with the IBM system, the error code is correct. If the error occurred while connecting to the IBM system, the DIGITAL SNA Access Server for Windows NT always returns SNALU62$_RESFRET and any sense code information is lost.

If an SNALU62$_RESFRET error code occurs while using the DIGITAL SNA Access Server for Windows NT, use the Microsoft Event Viewer to determine why the session activation failed. Refer to the Microsoft SNA Server and Event Viewer documentation for more information.

4.1.2 Valid Conversation State for SNALU62$ALLOCATE

SNALU62$ALLOCATE can only be issued when the conversation state is reset (that is, the conversation does not yet exist).

4.1.3 State Transition

When the return code is OK, the conversation enters the send state.

4.2 SNALU62$CONFIRM

The SNALU62$CONFIRM procedure sends a confirmation request to the remote IBM host transaction program and waits for a reply. The OpenVMS transaction program waits for the APPC/LU6.2 Programming Interface to return a reply to the status vector on completion. As a result of this procedure, the APPC/LU6.2 Programming Interface flushes its send buffer. This verb allows synchronization between the conversation partners.

Format:

status.wlc.v=SNALU62$CONFIRM (*resource_id.rlu.r*,
status_blk.wx.dx,
[rts_rec.wlu.r],
[efn.rlu.r],
[astadr.szem.r],
[astprm.rlu.r])

Arguments:

status When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits together represent the severity of the error. Returned by function value.

resource_id An ID assigned to the conversation by the APPC/LU6.2 Programming Interface during conversation allocation. Passed by reference.

status_blk A longword vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.

rts_rec A location to receive one of the following values indicating whether the remote IBM host transaction program has requested permission to send a message. When TRUE, the least significant bit is set. When FALSE, the least significant bit is clear.

- TRUE: The remote IBM host transaction program has requested that the OpenVMS transaction program enter receive state and thereby place the remote program in send state.
- FALSE: The remote IBM host transaction program has not requested that the OpenVMS transaction program enter receive state.

Passed by reference.

efn	An event flag to set when notification becomes due. If this parameter is omitted, event flag 0 is set. Passed by reference.
astadr	The address of an AST routine to call when notification becomes due. Passed by reference.
astprm	A parameter to pass to the AST routine when notification occurs. This AST routine is called with this parameter only. Passed by reference.

4.2.1 Status Codes

The SNALU62$CONFIRM procedure can return the following successful completion code:

- SNALU62$_OK

The SNALU62$CONFIRM procedure can return the following error codes:

- SNALU62$_ALLERR
- SNALU62$_DEABPR
- SNALU62$_DEABSVC
- SNALU62$_DEABTIM
- SNALU62$_FMHNOT
- SNALU62$_MAPEFAI
- SNALU62$_MAPNFOU
- SNALU62$_PARERR
- SNALU62$_PRERPU
- SNALU62$_RESFNO
- SNALU62$_RESFRET
- SNALU62$_STAERR
- SNALU62$_SVCERPU
- SNALU62$_UNSUC

4.2.2 Valid Conversation State for SNALU62$CONFIRM

The SNALU62$CONFIRM procedure can only be issued in the send state.

4.2.3 State Transition

When the return code indicates OK, there is no state transition.

4.3 SNALU62$CONFIRMED

The SNALU62$CONFIRMED procedure sends a confirmation reply to the remote IBM host transaction program. This verb allows the local and remote transaction programs to synchronize their processing. Call this procedure upon receipt of an SNALU62$CONFIRM request from the remote IBM host transaction program.

Format:

status.wlc.v=SNALU62$CONFIRMED (*resource_id.rlu.r*, *status_blk.wx.dx*)

Arguments:

status	When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits together represent the severity of the error. Returned by function value.
resource_id	An ID assigned to the conversation by the APPC/LU6.2 Programming Interface during conversation allocation. Passed by reference.
status_blk	A longword vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.

4.3.1 Status Codes

The SNALU62$CONFIRMED procedure returns the following successful completion code:

- SNALU62$_OK

The SNALU62$CONFIRMED procedure returns the following error codes:

- SNALU62$_PARERR
- SNALU62$_STAERR
- SNALU62$_UNSUC

4.3.2 Valid Conversation State for SNALU62$CONFIRMED

The SNALU62$CONFIRMED procedure can only be issued in the confirm state.

4.3.3 State Transition

The following state transitions can occur with the SNALU62$CONFIRMED procedure:

- The conversation enters receive state when CONFIRM was received on the preceding SNALU62$RECEIVE_AND_WAIT or SNALU62$RECEIVE_AND_WAIT procedure.
- The conversation enters send state when CONFIRM_SEND was received on the preceding SNALU62$RECEIVE_AND_WAIT or SNALU62$RECEIVE_IMMEDIATE procedure.
- The conversation enters deallocate state when CONFIRM_DEALLOCATE was received on the preceding SNALU62$RECEIVE_AND_WAIT or SNALU62$RECEIVE_IMMEDIATE procedure.

4.4 SNALU62$DEALLOCATE

The SNALU62$DEALLOCATE procedure deallocates a conversation between an OpenVMS transaction program and a remote IBM host transaction program. When the procedure completes, the conversation that has been using the session ends. The session, however, remains active and available for use by another conversation.

Format:

status.wlc.v=SNALU62$DEALLOCATE (*resource_id.rlu.r*,
status_blk.wx.dx,
[*type.rlu.r*],
[*log_data.rx.dx*],
[*efn.rlu.r*],
[*astadr.szem.r*],
[*astprm.rlu.r*])

Arguments:

status	When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits together represent the severity of the error. Returned by function value.
resource_id	An ID assigned to the conversation by the APPC/LU6.2 Programming Interface during conversation allocation. Passed by reference.
status_blk	A longword vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.

type

A value indicating the type of deallocation to be performed: sync level (default), flush, abend, or local. Local deallocation is performed after the partner transaction program has deallocated its end of the conversation (indicated by the result of a previous call). This type of deallocation frees resources (virtual memory) allocated when the conversation was allocated. The other deallocation types are used to initiate the deallocation of a conversation.

This parameter can take the following values:

- SNALU62$K_SYNC_LEVEL
  This is the default. The synchronization level specified for this conversation determines the way the deallocation is performed: If SYNC_LEVEL(NONE), flush the send buffer and deallocate the conversation normally. If SYNC_LEVEL(CONFIRM), execute the SNALU62$CONFIRM function and, if it is successful, deallocate the conversation. If it is not successful, the state of the conversation is determined by the return code.

- SNALU62$K_FLUSH
  Execute the function of the SNALU62$FLUSH procedure and deallocate the conversation normally.

- SNALU62$K_ABEND_PROG
  Intended to be used by a transaction program when an error condition is detected that prevents the successful completion of the transaction. (Program-defined)
  SNALU62$K_ABEND_SVC
  Intended to be used by an LU services component when it detects an error condition caused by its peer LU services component in the remote LU. (Product-defined)
  SNALU62$K_ABEND_TIMER
  Intended to be used by an LU services component when it detects or is informed of a condition that requires the conversation to be deallocated without further communications. (Product-defined)

  DIGITAL SNA Access Server for Windows NT Note:
  On mapped conversations using the DIGITAL SNA Access Server for Windows NT, all abend options are changed to the generic AP_ABEND option supported by the Microsoft SNA Server.

Note

Flush the send buffer and deallocate the conversation abnormally. Truncation of logical records can occur when the conversation is in the send state, if the buffer being flushed contains less than a full record. Data purging can occur when the conversation is in the receive

state and the APPC/LU6.2 Programming Interface buffer is flushed before the data is passed to the OpenVMS transaction program. These three abnormal deallocations are used for controlled crashing when catastrophic errors are detected.

- SNALU62$K_LOCAL
  Deallocate the conversation locally. You must specify this type of deallocation if and only if the conversation is in the deallocate state. The conversation enters the deallocate state when the OpenVMS transaction program receives a status code (returned as the result of a previous call) indicating that the conversation has been deallocated. This type of deallocation frees up local resources (virtual memory) that were allocated when the conversation was allocated.

Passed by reference.

log_data — Product specific error information to be placed in the system error logs of the LUs supporting this conversation. The log data is not seen by the partner transaction program. Passed by descriptor.

efn — An event flag to set when notification becomes due. If this parameter is omitted, event flag 0 is set. Passed by reference.

astadr — The address of an AST routine to call when notification becomes due. Passed by reference.

astprm — A parameter to pass to the AST routine when notification occurs. This AST routine is called with this parameter only. Passed by reference.

4.4.1 Status Codes

The SNALU62$DEALLOCATE procedure returns the following successful completion code:

- SNALU62$_OK

The SNALU62$DEALLOCATE procedure returns the following error codes:

- SNALU62$_ALLERR
- SNALU62$_DEABPR
- SNALU62$_DEABSVC
- SNALU62$_DEABTIM
- SNALU62$_FMHNOT

- SNALU62$_MAPEFAI
- SNALU62$_MAPNFOU
- SNALU62$_PARERR
- SNALU62$_PRERPU
- SNALU62$_RESFNO
- SNALU62$_RESFRET
- SNALU62$_STAERR
- SNALU62$_SVCERPU
- SNALU62$_UNSUC

4.4.2 Valid Conversation States for SNALU62$DEALLOCATE

The valid conversation states for SNALU62$DEALLOCATE are as follows:

- The valid conversation state for DEALLOCATE TYPE(FLUSH) and DEALLOCATE TYPE(SYNC_LEVEL) is send.
- The valid conversation states for DEALLOCATE TYPE(ABEND) are send, receive, and confirm.
- The valid conversation state for DEALLOCATE TYPE(LOCAL) is deallocate.

4.4.3 State Transition

When the return code indicates OK, the conversation enters the reset state (the conversation no longer exists).

4.5 SNALU62$FLUSH

The SNALU62$FLUSH procedure causes the APPC/LU6.2 Programming Interface to flush its send buffer and transmit the contents to the remote IBM host transaction program.

Format:

status.wlc.v=SNALU62$FLUSH (*resource_id.rlu.r*, *status_blk.wx.dx*)

Arguments:

status	When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits together represent the severity of the error. Returned by function value.
resource_id	An ID assigned to the conversation by the APPC/LU6.2 Programming Interface during conversation allocation. Passed by reference.
status_blk	A longword vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.

4.5.1 Status Codes

The SNALU62$FLUSH procedure returns the following successful completion code:

- SNALU62$_OK

The SNALU62$FLUSH procedure returns the following error codes:

- SNALU62$_PARERR
- SNALU62$_STAERR
- SNALU62$_UNSUC

4.5.2 Valid Conversation State for SNALU62$FLUSH

The valid conversation state for SNALU62$FLUSH is send.

4.5.3 State Transition

When the return code indicates OK, the conversation undergoes no state transition.

4.6 SNALU62$GET_ATTRIBUTES

The SNALU62$GET_ATTRIBUTES procedure returns information about the specified conversation.

Format:

status.wlc.v=SNALU62$GET_ATTRIBUTES(*resource_id.rlu.r*,
status_blk.wx.dx,
[*qualified_own_lu_name.wt.dx*],
[*partner_lu_name.wt.dx*],
[*qualified_partner_lu_name.wt.dx*],
[*mode_name.wt.dx*],
[*sync_level.wlu.r*],
[*security_user_id.wt.dx*],
[*security_profile.wt.dx*],
[*luw_identifier.wlu.r*],
[*conversation_correlator.wlu.r*],
[*conv_state.wlu.r*],
[*session_id.wlu.r*])

Arguments:

status	When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits together represent the severity of the error. Returned by function value.
resource_id	An ID assigned to the conversation by the APPC/LU6.2 Programming Interface during session allocation. Passed by reference.
status_blk	A longword vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.
qualified_own_lu_name	A location to receive the fully qualified name used by the SNA network to identify the local LU. The name is returned in ASCII. If the LU name is unknown, a null value is returned. Passed by descriptor.
partner_lu_name	A location to receive the locally known name of the remote LU. The name is returned in ASCII. Passed by descriptor.
qualified_partner_lu_name	A location to receive the fully qualified name used by the SNA network to identify the remote LU. The name is returned in ASCII. Passed by descriptor.

mode_name	A location to receive the logon mode entry name for the session allocated to the conversation. The name is returned in ASCII. Passed by descriptor.
sync_level	A location to receive one of the following values indicating the level of synchronization used by the conversation: • SNALU62$K_SL_NONE - The conversation is allocated with no synchronization. • SNALU62$K_SL_CONFIRM - The conversation is allocated with a synchronization level of CONFIRM. Passed by reference.
security_user_id	A location to receive the user ID carried on the allocation request. Passed by descriptor.
security_profile	A location to receive the profile carried on the allocation request. Passed by descriptor. **DIGITAL SNA Access Server for Windows NT Note:** This parameter is not returned by the DIGITAL SNA Access Server for Windows NT.
luw_identifier	Reserved for future use. This parameter is ignored. Passed by reference.
conversation_correlator	Reserved for future use. This parameter is ignored. Passed by reference.

conv_state

A location to receive a value indicating the current state of the conversation. Possible values are:

- SNALU62$K_STATE_RESET is the state in which the program can allocate the conversation.
- SNALU62$K_STATE_SEND is the state in which the program can send data, request confirmation, or request sync point.
- SNALU62$K_STATE_RECEIVE is the state in which the program can receive information from the remote program.
- SNALU62$K_STATE_CONFIRM is the state in which the program can reply to a confirmation request.
- SNALU62$K_STATE_CONFIRM_DEALLOC is the state in which the program can reply to a deallocate request.
- SNALU62$K_STATE_DEALLOCATE is the state in which the program can deallocate the conversation locally.

Passed by reference.

session_id

A location to receive an ID value assigned to the session by the APPC/LU6.2 Programming Interface. Passed by reference.

4.6.1 Status Codes

The SNALU62$GET_ATTRIBUTES procedure returns the following successful completion code:

- SNALU62$_OK

The SNALU62$GET_ATTRIBUTES procedure returns the following error codes:

- SNALU62$_PARERR
- SNALU62$_STAERR
- SNALU62$_UNSUC

4.6.2 Valid Conversation States for SNALU62$GET_ATTRIBUTES

SNALU62$GET_ATTRIBUTES is valid in all states except reset.

4.6.3 State Transition

None.

4.7 SNALU62$GET_PIP

The SNALU62$GET_PIP verb returns Program Initialization Parameters (PIPs) that were supplied by the remote TP when it initiated the conversation. The content of the PIP is application dependent; it has meaning only to the cooperating transaction programs. For example, the name of a database can be sent by the IBM transaction program to the OpenVMS transaction program as PIP information. Program initialization parameters are not acted upon or examined by the LU.

Format:

status.wlc.v=SNALU62$GET_PIP (*resource_id.rlu.r*,
status_blk.wx.dx
num_pip.wlu.r,
[*pip_01.wt.dx*],
[*pip_02.wt.dx*],
[*pip_03.wt.dx*],
.
.
.
[*pip_14.wt.dx*],
[*pip_15.wt.dx*],
[*pip_16.wt.dx*])

Arguments:

status — When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits together represent the severity of the error. Returned by function value.

resource_id — An ID assigned to the conversation by the APPC/LU6.2 Programming Interface during conversation allocation. Passed by descriptor.

status_blk — A longword vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.

num_pip — A value indicating the number of PIPs supplied with the call to this procedure. If the remote TP specifies 5 Program Initialization Parameters (PIPs), *num pip* will be set to 5. Passed by reference.

pip_01,...,pip_16 — Each *pip_01,...,pip_16* is a location containing individual program initialization data that was specified by the remote TP. Passed by descriptor.

4.7.1 Status Codes

The SNALU62$GET_PIP procedure returns the following successful completion code:

- SNALU62$_OK

The SNALU62$GET_PIP procedure returns the following error code:

- SNALU62$_UNSUC

4.7.2 Valid Conversation State for SNALU62$GET_PIP

SNALU62$GET_PIP cannot be used in reset or deallocate state.

4.7.3 State Transition

None.

4.8 SNALU62$GET_TYPE

The SNALU62$GET_TYPE procedure returns a value indicating whether the specified conversation is a mapped or a basic conversation.

Format:

status.wlc.v=SNALU62$GET_TYPE (*resource_id.rlu.r*,
status_blk.wx.dx,
[*type.wlu.r*])

Arguments:

status	When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits together represent the severity of the error. Returned by function value.
resource_id	An ID assigned to the conversation by the APPC/LU6.2 Programming Interface during session allocation. Passed by reference.
status_blk	A longword vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.
type	A variable to receive one of the following values indicating the conversation type: • SNALU62$K_BASIC_CONVERSATION • SNALU62$K_MAPPED_CONVERSATION Passed by reference.

4.8.1 Status Codes

The SNALU62$GET_TYPE procedure can return the following successful completion code:

- SNALU62$_OK

The SNALU62$GET_TYPE procedure returns the following error codes:

- SNALU62$_PARERR
- SNALU62$_STAERR
- SNALU62$_UNSUC

4.8.2 Valid Conversation States for SNALU62$GET_TYPE

The valid conversation states for SNALU62$GET_TYPE are send, receive, and confirm.

4.8.3 State Transition

None.

4.9 SNALU62$POST_ON_RECEIPT

The SNALU62$POST_ON_RECEIPT procedure causes the APPC/LU6.2 Programming Interface to notify the OpenVMS transaction program when data, conversation status, or a request for confirmation is received from the remote IBM host transaction program. Notification can be made by means of an event flag, an asynchronous system trap (AST), or SNALU62$WAIT (See Section 4.17). The data or status is input into a user buffer by means of the SNALU62$RECEIVE_AND_WAIT procedure.

Format:

status.wlc.v=SNALU62$POST_ON_RECEIPT (*resource_id.rlu.r*,
status_blk.wx.dx,
[*fill.rlu.r*],
[*length.rwu.r*],
[*efn.rlu.r*],
[*astadr.szem.r*],
[*astprm.rlu.r*])

Arguments:

status — When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits together represent the severity of the error. Returned by function value.

resource_id — An ID assigned to the conversation by the APPC/LU6.2 Programming Interface during session allocation. Passed by reference.

status_blk — A longword vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.

fill

A value specifying when notification is to occur. For mapped conversations, only SNALU62$K_LL is valid. The parameter can take one of two values:

- SNALU62$K_BUFFER
  Notify when data is available whose length is at least equal to that specified on the *length* parameter, or when the end of data is available.
- SNALU62$K_LL
  Notify when a complete logical record or the remainder of a previously truncated logical record is available, or when part of a logical record is available whose length is at least equal to that specified on the *length* parameter, whichever occurs first. This value is the default.

Passed by reference.

length

Used with the fill parameter. A value specifying the maximum length of logical record data that the OpenVMS transaction program can receive before notification.

- If *fill* SNALU62$K_BUFFER is specified, the program is notified when a complete message is received.
- If *fill* SNALU62$K_LL is specified, the program is notified when a complete logical record is received.

Passed by reference.

efn

An event flag to set when notification becomes due. If this parameter is omitted, event flag 0 is set. Passed by reference.

astadr

The address of an AST routine to call when notification becomes due. Passed by reference.

astprm

A parameter to pass to the AST routine when notification occurs. This AST routine is called with this parameter only. Passed by reference. When notification becomes due, the user-written AST procedure is called as follows:

ast_procedure (user_prm.rlu.r)

where

ast_procedure

is the name of the user's routine that is being called. (*ast_procedure* is specified as the *astadr* parameter in the SNALU62$POST_ON_RECEIPT procedure call.)

user_prm is a parameter passed to the user-written routine. (*user_prm* is specified as *astprm* in the SNALU62$POST_ON_RECEIPT procedure.)

4.9.1 Status Codes

The SNALU62$POST_ON_RECEIPT procedure can return the following successful completion code:

- SNALU62$_OK

The SNALU62$POST_ON_RECEIPT procedure returns the following error codes:

- SNALU62$_PARERR
- SNALU62$_STAERR
- SNALU62$_UNSUC

4.9.2 Valid Conversation State for SNALU62$POST_ON_RECEIPT

The valid conversation state for SNALU62$POST_ON_RECEIPT is receive.

4.9.3 State Transition

None.

4.10 SNALU62$PREPARE_TO_RECEIVE

The SNALU62$PREPARE_TO_RECEIVE procedure changes the conversation from send to receive in preparation for receiving data from the remote IBM host transaction program. This procedure includes the function of the SNALU62$FLUSH or SNALU62$CONFIRM procedure. For additional information, see Appendix D.

Format:

status.wlc.v=SNALU62$PREPARE_TO_RECEIVE (*resource_id.rlu.r*,
status_blk.wx.dx,
[*type.rlu.r*],
[*locks.rlu.r*],
[*efn.rlu.r*],
[*astadr.szem.r*],
[*astprm.rlu.r*])

Arguments:

status	When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits together represent the severity of the error. Returned by function value.
resource_id	An ID assigned to the conversation by the APPC/LU6.2 Programming Interface during conversation allocation. This ID must be used on all subsequent calls for this conversation. Passed by reference.
status_blk	A longword vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.

type

A value specifying the type of SNALU62$PREPARE_TO_RECEIVE to be performed for this conversation. This parameter can take the following values:

- SNALU62$K_SYNC_LEVEL
  This is the default. The synchronization level of the conversation determines the effect of the SNALU62$PREPARE_TO_RECEIVE function: If SYNC_LEVEL(NONE), execute the SNALU62$FLUSH function and enter the receive state. If SYNC_LEVEL(CONFIRM), execute the SNALU62$CONFIRM function and, if successful, enter the receive state. If the SNALU62$CONFIRM is not successful, the state of the conversation is determined by the return code (see Appendix C).
- SNALU62$K_FLUSH
  Execute the SNALU62$FLUSH function and enter the receive state.

Passed by reference.

locks

A value that indicates when control is to be returned to the OpenVMS transaction program. Valid only when the *type* parameter specifies SNALU62$K_SYNC_LEVEL and the conversation is using SNALU62$CONFIRM processing. The parameter can have the following values:

- SNALU62$K_SHORT
  Return control when an affirmative reply is received. This is the default. If the synchronization level for the conversation is CONFIRM, return control when an SNALU62$CONFIRMED reply is received.
- SNALU62$K_LONG
  Return control when information such as data is received from the remote IBM host transaction program following an affirmative reply. If the synchronization level is CONFIRM, return control when information is received following an SNALU62$CONFIRMED reply.

Passed by reference.

efn

An event flag to set when notification becomes due. If this parameter is omitted, event flag 0 is set. Passed by reference.

astadr

The address of an AST routine to call when notification becomes due. Passed by reference.

astprm

A parameter to pass to the AST routine when notification occurs. This AST routine is called with this parameter only. Passed by reference.

4.10.1 Status Codes

The SNALU62$PREPARE_TO_RECEIVE procedure returns the following successful completion code:

- SNALU62$_OK

The SNALU62$PREPARE_TO_RECEIVE procedure returns the following error codes:

- SNALU62$_ALLERR
- SNALU62$_DEABPR
- SNALU62$_DEABSVC
- SNALU62$_DEABTIM
- SNALU62$_DEALNOR
- SNALU62$_FMHNOT
- SNALU62$_MAPEFAI
- SNALU62$_MAPNFOU
- SNALU62$_PARERR
- SNALU62$_PRERPU
- SNALU62$_RESFNO
- SNALU62$_RESFRET
- SNALU62$_STAERR
- SNALU62$_SVCERPU
- SNALU62$_UNSUC

4.10.2 Valid Conversation State for SNALU62$PREPARE_TO_RECEIVE

The valid conversation state for SNALU62$PREPARE_TO_RECEIVE is send.

4.10.3 State Transition

When the return code indicates OK, the conversation enters receive state when TYPE(FLUSH) is specified, or when TYPE(SYNC_LEVEL) is specified and the synchronization level is NONE or CONFIRM.

4.11 SNALU62$RECEIVE_AND_WAIT

The SNALU62$RECEIVE_AND_WAIT procedure waits for information to arrive on a conversation from the remote IBM host transaction program and receives it. When this occurs, the APPC/LU6.2 Programming Interface passes the message to the OpenVMS transaction program and the procedure completes. The information may be data, conversation status, or a request for confirmation.

If the OpenVMS transaction program calls the procedure while in the send state, the APPC/LU6.2 Programming Interface flushes the send buffer and the OpenVMS transaction program goes into the receive state.

For a basic conversation, the *fill* parameter determines the format of the data received. For a mapped conversation, data is received one record at a time, and it does not contain the LL header.

Format:

status.wlc.v=SNALU62$RECEIVE_AND_WAIT (*resource_id.rlu.r*,
status_blk.wx.dx,
[*fill.rlu.r*],
[*length.wwu.r*],
[*rts_rec.wlu.r*],
[*data.wx.dx*],
[*what_received.wlu.r*],
[*map_name.wx.dx*],
[*efn.rlu.r*],
[*astadr.szem.r*],
[*astprm.rlu.r*])

Arguments:

status When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits together represent the severity of the error. Returned by function value.

resource_id An ID assigned to the conversation by the APPC/LU6.2 Programming Interface during session allocation. Passed by reference.

status_blk A longword vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.

fill

A value indicating whether the data is received in logical-record format or independent of logical-record format.

- SNALU62$K_BUFFER
  Fill the buffer equal to the length specified in the buffer descriptor or until the end of the data is detected independent of the logical record format. For D-type descriptors, data is received until the end of the data is detected. Not valid for mapped conversations.
- SNALU62$K_LL
  Receive a complete or truncated logical record up to the length specified in the buffer descriptor. This is the default. For class D descriptors, a complete logical record is received.

This parameter only has meaning for basic conversations. Passed by reference.

length

A location to receive the actual length of the data received. Updated only if the received message is data. Passed by reference.

rts_rec

A location to receive one of the following values indicating whether the remote IBM host transaction program has requested permission to send a message. When TRUE, the least significant bit is set. When FALSE, the least significant bit is clear.

- TRUE: The remote IBM host transaction program has requested that the OpenVMS transaction program enter receive state and thereby place the remote program in send state.
- FALSE: The remote IBM host transaction program has not requested that the OpenVMS transaction program enter receive state.

Passed by reference.

Note

Unless SNALU62$K_SEND or SNALU62$K_CONFIRM_SEND is returned in the *what_received* parameter on this verb, the OpenVMS transaction program must issue another SNALU62$K_RECEIVE_AND_WAIT call in order to transition into the receive state. The setting of the *rts_rec* parameter has no effect on the resultant state of the conversation.

data

A buffer to receive data from the remote IBM host transaction program. Data is moved to the buffer depending on the values indicated in the *fill* and *what_received* parameters. The parameter gives the total size of the buffer area; it should be large enough to contain the largest record expected. The actual length filled is returned in the *length* parameter. For a basic conversation:

- If the *fill* parameter equals SNALU62$K_LL, the buffer contains a 2-byte length field (LL) followed by a data field (see Figure 2–1).
- If the *fill* parameter equals SNALU62$K_BUFFER, the buffer potentially contains multiple records, each consisting of a 2-byte length field followed by a data field. The last record may be truncated.

For a mapped conversation, only one record is returned and it does not contain the LL header (see Figure 2–2).

The *data* parameter is passed by descriptor. If the descriptor is dynamic, the buffer will be allocated by the interface and returned to the user.

what_received

A location to receive a value indicating the type of message received. This location can contain the following values:

- SNALU62$K_DATA
  A value returned when *fill* = SNALU62$K_BUFFER is specified, and data has been received by the OpenVMS transaction program. In a basic conversation, this indicates that data has been received independent of the logical record structure.
- SNALU62$K_DATA_COMPLETE
  A value returned when *fill* = SNALU62$K_LL is specified, and a complete logical record has been received. In a basic conversation, this indicates that a complete logical record has been received.
- SNALU62$K_DATA_INCOMPLETE
  A value returned when *fill* = SNALU62$K_LL is specified, and less than a complete logical record has been received. In a mapped conversation, this indicates that less than a complete data record has been received.
- SNALU62$K_LL_TRUNCATED
  A value returned when *fill* = SNALU62$K_LL is specified, and the 2-byte LL field of a logical record has been truncated after the first byte.
- SNALU62$K_FMH_DATA_COMPLETE
  A value that indicates a complete data record or the last remaining portion thereof is received by the program and the data record contains FM headers. This value can only be returned for mapped conversations. In a mapped conversation, this indicates that less than a complete data record has been received.
- SNALU62$K_FMH_DATA_INCOMPLETE
  In a mapped conversation, this indicates that less than a complete logical record plus function management header has been received.

- SNALU62$K_SEND
  A value that indicates that the remote IBM host transaction program has entered receive state, putting the OpenVMS transaction program in the send state. The OpenVMS transaction program may now call the SNALU62$SEND_DATA procedure.
- SNALU62$K_CONFIRM
  A value that indicates if the remote IBM host transaction program has issued an SNALU62$CONFIRM to force a synchronization point. The OpenVMS transaction program must respond by issuing SNALU62$CONFIRMED or SNALU62$SEND_ERROR.
- SNALU62$K_CONFIRM_SEND
  A value indicating that the remote IBM host transaction program has issued SNALU62$PREPARE_TO_RECEIVE with type SYNC_LEVEL and the synchronization level is CONFIRM. The OpenVMS transaction program can respond by calling the SNALU62$CONFIRMED procedure or another procedure, such as SNALU62$SEND_ERROR.
- SNALU62$K_CONFIRM_DEALLOCATE
  A value indicating that the remote IBM host transaction program has issued SNALU62$DEALLOCATE with type SYNC_LEVEL and the synchronization level is CONFIRM. The OpenVMS transaction program can respond by calling SNALU62$CONFIRMED or by calling another procedure, such as SNALU62$SEND_ERROR.

This call results in either data or a single status indicator. Data is never received together with a status indicator. Passed by reference.

map_name Reserved for future use. This parameter is ignored. Passed by descriptor.

efn An event flag to set when notification becomes due. If this parameter is omitted, event flag 0 is set. Passed by reference.

astadr The address of an AST routine to call when notification becomes due. Passed by reference.

astprm A parameter to pass to the AST routine when notification occurs. This AST routine is called with this parameter only. Passed by reference.

4.11.1 Status Codes

The SNALU62$RECEIVE_AND_WAIT procedure can return the following successful completion code:

- SNALU62$_OK

The SNALU62$RECEIVE_AND_WAIT procedure returns the following error codes:

- SNALU62$_ALLERR

- SNALU62$_DEABPR
- SNALU62$_DEABSVC
- SNALU62$_DEABTIM
- SNALU62$_DEALNOR
- SNALU62$_FMHNOT
- SNALU62$_MAPEFAI
- SNALU62$_MAPNFOU
- SNALU62$_PARERR
- SNALU62$_PRERNTR
- SNALU62$_PRERPU
- SNALU62$_PRERTR
- SNALU62$_RESFNO
- SNALU62$_RESFRET
- SNALU62$_STAERR
- SNALU62$_SVCENTR
- SNALU62$_SVCERPU
- SNALU62$_SVCERTR
- SNALU62$_UNSUC

4.11.2 Valid Conversation States for SNALU62$RECEIVE_AND_WAIT

The valid conversation states for SNALU62$RECEIVE_AND_WAIT are send and receive.

4.11.3 State Transition

When the return code indicates OK:

- The conversation enters receive state when the procedure is issued in send state and when the *what_received* parameter indicates DATA, DATA_COMPLETE, DATA_INCOMPLETE, FMH_DATA_COMPLETE, or FMH_DATA_INCOMPLETE.
- The conversation enters send state when the *what_received* parameter indicates send.

- The conversation enters confirm state when the *what_received* parameter indicates CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE.
- The conversation does not undergo a state change when the procedure is issued in receive state and when *what_received* indicates DATA, DATA_COMPLETE, DATA_INCOMPLETE, FMH_DATA_COMPLETE, or FMH_DATA_INCOMPLETE.

4.12 SNALU62$RECEIVE_IMMEDIATE

The SNALU62$RECEIVE_IMMEDIATE procedure checks to see if information has arrived from the remote IBM host transaction program. The procedure can only be called in the receive state. The information may be data, conversation status, or a request for confirmation. If the procedure returns the status code SNALU62$_UNSUC, with the suberror SNALU62$_NDAVAIL (no data available), then either insufficient data or no data transferred into the user's buffer.

For a basic conversation, the *fill* parameter determines the format of the data received. For a mapped conversation, data is received one record at a time.

Format:

status.wlc.v=SNALU62$RECEIVE_IMMEDIATE (*resource_id.rlu.r*,
status_blk.wx.dx,
[*fill.rlu.r*],
[*length.wwu.r*],
[*rts_rec.wlu.r*],
[*data.wx.dx*],
[*what_received.wlu.r*],
[*map_name.wx.dx*])

Arguments:

status — When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits together represent the severity of the error. Returned by function value.

resource_id — An ID assigned to the conversation by the APPC/LU6.2 Programming Interface during session allocation. Passed by reference.

status_blk — A longword vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.

fill

A value indicating whether the data is to be received in logical-record format or independent of logical-record format.

- SNALU62$K_BUFFER
  Fill the buffer equal to the length specified in the buffer descriptor or until the end of the data is detected independent of the logical record format. For D-type descriptors, data is received until the end of the data is detected.
- SNALU62$K_LL
  Receive a complete or truncated logical record up to the length specified in the buffer descriptor. This is the default. For class D descriptors, a complete logical record is received.

This parameter only has meaning for basic conversations. Passed by reference.

length

A location to receive the actual length of the data received. Updated only if the received message is data. Passed by reference.

rts_rec

A location to receive one of the following values indicating whether the remote IBM host transaction program has requested permission to send a message. When TRUE, the least significant bit is set. When FALSE, the least significant bit is clear.

- TRUE: The remote IBM host transaction program has requested that the OpenVMS transaction program enter receive state and thereby place the remote program in send state.
- FALSE: The remote IBM host transaction program has not requested that the OpenVMS transaction program enter receive state.

Passed by reference.

data

A buffer to receive data from the remote IBM host transaction program. Data is moved to the buffer depending on the values indicated in the *fill* and *what_received* parameters. The parameter gives the total size of the buffer area; it should be large enough to contain the largest record expected. The actual length filled is returned in the *length* parameter. For a basic conversation:

- If the *fill* parameter equals SNALU62$K_LL, the buffer contains a 2-byte length field (LL) followed by a data field (see Figure 2–1).
- If the *fill* parameter equals SNALU62$K_BUFFER, the buffer potentially contains multiple records, each consisting of a 2-byte length field followed by a data field. The last record may be truncated.

For a mapped conversation, only one record is returned and it does not contain the LL header (see Figure 2–2). Passed by descriptor.

what_received

A location to receive a value indicating the type of message received. This location can contain the following values:

- SNALU62$K_DATA
  A value returned when *fill* = SNALU62$K_BUFFER is specified, and data has been received by the OpenVMS transaction program. In a basic conversation, this indicates that data has been received independent of the logical record structure.
- SNALU62$K_DATA_COMPLETE
  A value returned when *fill* = SNALU62$K_LL is specified, and a complete logical record has been received. In a basic conversation, this indicates that a complete logical record has been received. In a mapped conversation, this indicates that a complete data record or the last part of a record has been received.
- SNALU62$K_DATA_INCOMPLETE
  A value returned when *fill* = SNALU62$K_LL is specified, and less than a complete logical record has been received. In a mapped conversation, this indicates that less than a complete data record has been received.
- SNALU62$K_LL_TRUNCATED
  A value returned when *fill* = SNALU62$K_LL is specified, and the 2-byte LL field of a logical record has been truncated after the first byte.
- SNALU62$K_FMH_DATA_COMPLETE
  A value that indicates a complete data record or its last remaining portion is received by the program and the data record contains FM headers. This value can only be returned for mapped conversations.
- SNALU62$K_FMH_DATA_INCOMPLETE
  In a mapped conversation, this indicates that less than a complete logical record plus function management header has been received.

- SNALU62$K_SEND
  A value that indicates that the remote IBM host transaction program has entered receive state, putting the OpenVMS transaction program in the send state. The OpenVMS transaction program may now call the SNALU62$SEND_DATA procedure.
- SNALU62$K_CONFIRM
  A value that indicates if the remote IBM host transaction program has issued an SNALU62$CONFIRM to force a synchronization point. The OpenVMS transaction program must respond by issuing SNALU62$CONFIRMED or SNALU62$SEND_ERROR.
- SNALU62$K_CONFIRM_SEND
  A value indicating that the remote IBM host transaction program has issued SNALU62$PREPARE_TO_RECEIVE with type SYNC_LEVEL, and synchronization level is CONFIRM. The OpenVMS transaction program can respond by calling the SNALU62$CONFIRMED procedure or another procedure, such as SNALU62$SEND_ERROR.
- SNALU62$K_CONFIRM_DEALLOCATE
  A value indicating that the remote IBM host transaction program has issued SNALU62$DEALLOCATE with type SYNC_LEVEL, and the synchronization level is CONFIRM. The OpenVMS transaction program can respond by calling SNALU62$CONFIRMED or by calling another procedure, such as SNALU62$SEND_ERROR.

This call results in either data or a single status indicator. Data is never received together with a status indicator. Passed by reference.

map_name — Reserved for future use. This parameter is ignored. Passed by descriptor.

4.12.1 Status Codes

The SNALU62$RECEIVE_IMMEDIATE procedure can return the following successful completion code:

- SNALU62$_OK

The SNALU62$RECEIVE_IMMEDIATE procedure returns the following error codes:

- SNALU62$_DEABPR
- SNALU62$_DEABSVC

- SNALU62$_DEABTIM
- SNALU62$_DEALNOR
- SNALU62$_FMHNOT
- SNALU62$_MAPEFAI
- SNALU62$_MAPNFOU
- SNALU62$_NDAVAIL
- SNALU62$_PARERR
- SNALU62$_PRERNTR
- SNALU62$_PRERPU
- SNALU62$_PRERTR
- SNALU62$_RESFNO
- SNALU62$_RESFRET
- SNALU62$_STAERR
- SNALU62$_SVCENTR
- SNALU62$_SVCERPU
- SNALU62$_SVCERTR
- SNALU62$_UNSUC

4.12.2 Valid Conversation State for SNALU62$RECEIVE_IMMEDIATE

The valid conversation state for SNALU62$RECEIVE_IMMEDIATE is receive.

4.12.3 State Transition

The conversation does not undergo a state change when the procedure is issued in receive state and when *what received* indicates DATA, DATA_COMPLETE, DATA_INCOMPLETE, FMH_DATA_COMPLETE, or FMH_DATA_INCOMPLETE.

4.13 SNALU62$REQUEST_TO_SEND

The SNALU62$REQUEST_TO_SEND procedure signals the remote IBM host transaction program that the OpenVMS transaction program wants to send data. The conversation will be changed to the send state when the OpenVMS transaction program subsequently receives a SEND indication from the remote IBM host transaction program by means of the SNALU62$RECEIVE_AND_WAIT procedure.

Format:

status.wlc.v=SNALU62$REQUEST_TO_SEND (*resource_id.rlu.r*, *status_blk.wx.dx*)

Arguments:

status	When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits together represent the severity of the error. Returned by function value.
resource_id	An ID assigned by the APPC/LU6.2 Programming Interface to the conversation. Passed by reference.
status_blk	A longword vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.

4.13.1 Status Codes

The SNALU62$REQUEST_TO_SEND procedure can return the following successful completion code:

- SNALU62$_OK

The SNALU62$REQUEST_TO_SEND procedure can return the following error codes:

- SNALU62$_PARERR
- SNALU62$_STAERR
- SNALU62$_UNSUC

4.13.2 Valid Conversation States for SNALU62$REQUEST_TO_SEND

The valid conversation states for SNALU62$REQUEST_TO_SEND are receive and confirm.

4.13.3 State Transition

None.

4.14 SNALU62$SEND_DATA

The SNALU62$SEND_DATA procedure sends data to the remote IBM host transaction program. The format of the data depends on the type of conversation being used:

- In a basic conversation, the OpenVMS transaction program specifies a user-supplied send buffer that contains one or more logical records. Each record is preceded by a 2-byte length field (LL) to indicate the length of the record including the 2-byte prefix (see Section 2.1.2.2.
- In a mapped conversation, the OpenVMS transaction program specifies a user-supplied send buffer that contains a single data record without the length field. The data can include a function management header.

Format:

status.wlc.v=SNALU62$SEND_DATA (*resource_id.rlu.r*,
status_blk.wx.dx,
data.rx.dx,
[*length.rwu.r*],
[*rts_rec.wlu.r*],
[*map_name.rt.dx*],
[*fmh_data.rlu.r*],
[*efn.rlu.r*],
[*astadr.szem.r*],
[*astprm.rlu.r*])

Arguments:

status	When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits together represent the severity of the error. Returned by function value.
resource_id	An ID assigned by the APPC/LU6.2 Programming Interface to the conversation. Passed by reference.
status_blk	A longword vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.
data	A buffer containing the data to be sent to the remote IBM host transaction program. Passed by descriptor.

length

A value indicating the length of the data to send. If the value is 0 or is omitted, *length* is derived from the length field in the *data* parameter descriptor. In a basic conversation, *length* is the length of the combined logical records in the buffer. Note that this is not the same as the value in the 2-byte length field prefix, which indicates the size of one logical record. In a mapped conversation, *length* is the length of the single data record in the buffer. Passed by reference.

rts_rec

A location to receive one longword of the following values indicating whether the remote IBM host transaction program has requested permission to send a message. When TRUE, the least significant bit is set. When FALSE, the least significant bit is clear.

- TRUE: The remote IBM host transaction program has requested that the OpenVMS transaction program enter receive state and so place the remote program in send state.
- FALSE: The remote IBM host transaction program has not requested that the OpenVMS transaction program enter receive state.

Passed by reference.

map_name

Reserved for future use. This parameter is ignored. Passed by descriptor.

fmh_data

Valid for a mapped conversation only. A flag indicating whether the data record includes a function management header. If this variable tests true, the data record includes an FM header. If this variable tests false, the data record does not include FM headers. If this parameter is supplied for a basic conversation, the error SNALU62$_PARERR is returned. Passed by reference.

efn

An event flag to set when notification becomes due. If this parameter is omitted, event flag 0 is set. Passed by reference.

astadr

The address of an AST routine to call when notification becomes due. Passed by reference.

astprm

A parameter to pass to the AST routine when notification occurs. This AST routine is called with this parameter only. Passed by reference.

4.14.1 Status Codes

The SNALU62$SEND_DATA procedure can return the following successful completion code:

- SNALU62$_OK

The SNALU62$SEND_DATA procedure can return the following status codes:

- SNALU62$_ALLERR
- SNALU62$_DEABPR
- SNALU62$_DEABSVC
- SNALU62$_DEABTIM
- SNALU62$_FMHNOT
- SNALU62$_MAPEFAI
- SNALU62$_MAPNFOU
- SNALU62$_PARERR
- SNALU62$_PRERPU
- SNALU62$_RESFNO
- SNALU62$_RESFRET
- SNALU62$_STAERR
- SNALU62$_SVCERPU
- SNALU62$_UNSUC

4.14.2 Valid Conversation State for SNALU62$SEND_DATA

The valid conversation state for SNALU62$SEND_DATA is send.

4.14.3 State Transition

When the return code indicates OK: none.

4.15 SNALU62$SEND_ERROR

The SNALU62$SEND_ERROR procedure informs the remote IBM host transaction program that the OpenVMS transaction program has detected an error. The APPC/LU6.2 Programming Interface flushes the send buffer if the OpenVMS transaction program is in the send state. Upon completion, the OpenVMS transaction program is in the send state and the remote IBM host transaction program is in the receive state.

Format:

status.wlc.v=SNALU62$SEND_ERROR (*resource_id.rlu.r*,
status_blk.wx.dx,
[*type.rlu.r*],
[*log_data.rx.dx*],
[*rts_rec.wlu.r*],
[*efn.rlu.r*],
[*astadr.szem.r*],
[*astprm.rlu.r*])

Arguments:

status
When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits together represent the severity of the error. Returned by function value.

resource_id
An ID assigned by the APPC/LU6.2 Programming Interface to the conversation during allocation. Passed by reference.

status_blk
A longword vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.

type
A value indicating the type of error being reported. The OpenVMS transaction program can specify either value. This parameter can take the following values:

- SNALU62$K_PROG
  Indicates an end-user transaction program error. This is the default.
- SNALU62$K_SVC
  Indicates an LU services error.

Passed by reference.

log_data	Product-specific error information that is to be placed in the system error logs of the LUs supporting this conversation. Passed by descriptor.
rts_rec	A location to receive one of the following values indicating whether the remote IBM host transaction program has requested permission to send a message. When TRUE, the least significant bit is set. When FALSE, the least significant bit is clear. • TRUE: The remote IBM host transaction program has requested that the OpenVMS transaction program enter receive state and so place the remote program in send state. • FALSE: The remote IBM host transaction program has not requested that the OpenVMS transaction program enter receive state. Passed by reference.
efn	An event flag to set when notification becomes due. If this parameter is omitted, event flag 0 is set. Passed by reference.
astadr	The address of an AST routine to call when notification becomes due. Passed by reference.
astprm	A parameter to pass to the AST routine when notification occurs. This AST routine is called with this parameter only. Passed by reference.

4.15.1 Status Codes

The SNALU62$SEND_ERROR procedure can return the following successful completion code:

- SNALU62$_OK

The SNALU62$SEND_ERROR procedure can return the following status codes:

- SNALU62$_ALLERR
- SNALU62$_DEABPR
- SNALU62$_DEABSVC
- SNALU62$_DEABTIM
- SNALU62$_DEALNOR
- SNALU62$_FMHNOT
- SNALU62$_MAPEFAI
- SNALU62$_MAPNFOU
- SNALU62$_PARERR

- SNALU62$_PRERPU
- SNALU62$_RESFNO
- SNALU62$_RESFRET
- SNALU62$_STAERR
- SNALU62$_SVCERPU
- SNALU62$_UNSUC

4.15.2 Valid Conversation States for SNALU62$SEND_ERROR

The valid conversation states for SNALU62$SEND_ERROR are send, receive, and confirm.

4.15.3 State Transition

When the return code indicates OK:

- The conversation enters send state when the procedure call is issued in receive or confirm state.
- The conversation does not undergo a state change when the procedure call is issued in send state.

4.16 SNALU62$SUPPLY_PIP

The SNALU62$SUPPLY_PIP verb defines Program Initialization Parameters that can be sent to a remote transaction program on a subsequent SNALU62$ALLOCATE verb, when a conversation is allocated by the OpenVMS transaction program.

Format:

status.wlc.v=SNALU62$SUPPLY_PIP (*status_blk.wx.dx*,
pip_context.wlu.r,
num_pip.rlu.r,
[*pip_01.rt.dx*],
[*pip_02.rt.dx*],
[*pip_03.rt.dx*],
.
.
.
[*pip_14.rt.dx*],
[*pip_15.rt.dx*],
[*pip_16.rt.dx*])

Arguments:

status — When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits together represent the severity of the error. Returned by function value.

status_blk — A longword vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.

pip_context — *pip_context* is a context variable that is returned by this procedure. This variable is supplied on a subsequent SNALU62$ALLOCATE call to actually supply the PIP parameters to the remote transaction program. Passed by reference.

num_pip — A value indicating the number of PIPs supplied with the call to this procedure. If you specify 5 PIPs, then locations for 5 PIPs are created and sent on a subsequent call to the SNALU62$ALLOCATE procedure. Passed by reference.

pip_01,...,pip_16 — Each *pip_01,...,pip_16* is a parameter containing program initialization data to send to the remote transaction program. Passed by descriptor.

4.16.1 Status Codes

The SNALU62$SUPPLY_PIP procedure returns the following successful completion code:

- SNALU62$_OK

The SNALU62$SUPPLY_PIP procedure returns the following error code:

- SNALU62$_UNSUC

4.16.2 Valid Conversation States for SNALU62$SUPPLY_PIP

Valid in all states.

4.16.3 State Transition

None.

4.17 SNALU62$WAIT

The SNALU62$WAIT procedure suspends execution of the OpenVMS transaction program until data, status, or a confirmation request is received on one of the specified conversations. These conversations must be set up for asynchronous completion by means of the SNALU62$POST_ON_RECEIPT procedure.

Format:

status.wlc.v=SNALU62$WAIT	(*resource_posted.wlu.r*, *status_blk.wx.dx*, *resource_list.rx.dx*, [*number_ids.rlu.r*])

Arguments:

status	When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits together represent the severity of the error. Returned by function value.
resource_posted	A longword to receive the resource_id of the conversation that has received data, status, or confirmation request. Passed by reference.
status_blk	A longword vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.
resource_list	An array of longwords specifying which conversations to monitor for receive completion. Each longword contains the resource ID of the conversation to be monitored. Other conversations will be ignored by this procedure. Passed by descriptor.
number_ids	A longword containing the number of conversations listed in the *resource_list*. If this parameter is omitted, the array size from the *resource_list* descriptor is used. Passed by reference.

4.17.1 Status Codes

The SNALU62$WAIT procedure can return the following status codes:

- SNALU62$_UNSUC

4.17.2 Valid Conversation State for SNALU62$WAIT

The valid conversation state for SNALU62$WAIT is receive.

4.17.3 State Transition

None.

5

Procedure Calling Format: Control Operator Verbs

This chapter describes the calling format, status codes, and state transitions for control operator procedures. The following procedures are available:

- SNALU62$ACTIVATE_SESSION
- SNALU62$DEACTIVATE_SESSION
- SNALU62$DEFINE_REMOTE
- SNALU62$DEFINE_TP
- SNALU62$DELETE

Calls to the APPC/LU6.2 Programming Interface procedures have the following general format:

status=SNALU62$procedure name (argument,...,[argument])

where

status	is a status code returned as a function value.
procedure_name	is the procedure you want to call.
( )	delimits the argument list.
argument	is a symbol containing information that the OpenVMS transaction program passes to the Programming Interface. Shorthand notation describes the argument's characteristics. Appendix A summarizes the notation.
[*argument*]	indicates an optional argument.

Arguments pass to the APPC/LU6.2 Programming Interface in two ways:

- By reference (or address). The argument is the address of an area or field that contains the value. An argument passed by address is usually expressed as a reference name or label associated with an area or field.

- By descriptor. This argument is also an address—for a special data structure called a descriptor (see the *Introduction to OpenVMS System Routines*).

In this chapter, the argument definitions for each procedure specify how each argument is passed.

Sessions are activated as the result of an OpenVMS transaction request for a conversation. The SNALU62$ACTIVATE_SESSION control operator verb activates a session.

You can define, modify and delete the local LU's operating parameters with the following verbs:

- SNALU62$DEFINE_REMOTE
- SNALU62$DEFINE_TP
- SNALU62$DELETE

5.1 SNALU62$ACTIVATE_SESSION

The SNALU62$ACTIVATE_SESSION procedure activates an LU-LU session, and specifies a mode name for the remote LU. The local LU may be either the contention winner or contention loser depending on the Bind Request received from the PLU and the requested polarity.

Format:

status.wlc.v=SNALU62$ACTIVATE_SESSION (*session_id.wlu.r*,
status_blk.wx.dx,
lu_name.rt.dx,
[*mode_name.rt.dx*],
[*efn.rlu.r*],
[*astadr.szem.r*],
[*astprm.rlu.r*],
[*notify_rtn.szem.r*],
[*notify_prm.rlu.r*],
[*notify_blk.wx.dx*],
[*polarity.rlu.r*])

Arguments:

status
When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits, together, represent the severity of the error. Returned by function value.

session_id
A location to receive an ID value assigned to the session by the APPC/LU6.2 Programming Interface. Passed by reference.

status_blk
A longword status vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.

lu_name
A value specifying the name of the remote LU with which the session is to be activated. This name must be a locally known LU name defined using the SNALU62$DEFINE_ REMOTE verb. Passed by descriptor.

mode_name
A logon mode table name associated with a set of BIND request parameters for the session. Passed by descriptor.

DIGITAL SNA Access Server for Windows NT Note:
This parameter must reference an APPC mode name defined on the Microsoft SNA Server.

efn	An event flag to set when notification becomes due. If this parameter is omitted, event flag 0 is set. Passed by reference.
astadr	The address of an AST routine to call when notification becomes due. Passed by reference.
astprm	A parameter to pass to the AST routine when notification occurs. This AST routine is called with this parameter only. Passed by reference.
notify_rtn	Notify which service routine to execute when the session activated by the procedure fails with no outstanding verb. Passed by reference.
notify_prm	Notify which parameter to pass to the notify service routine specified by the *notify_rtn* parameter. The user-written notify routine has the following calling format: *notify_rtn(notify_prm.rlu.r)*. Passed by reference.
notify_blk	A longword status vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface with complete status information regarding the session failure. Passed by descriptor.
polarity	Specifies the contention polarity for the session. SNALU62$K_FIRST_SPEAKER indicates that the local LU will be the contention winner. SNALU62$K_BIDDER indicates that the local LU will be the contention loser. SNALU62$K_FIRST_SPEAKER is the default value. Passed by reference.

5.1.1 Status Codes

The SNALU62$ACTIVATE_SESSION procedure returns the following successful status codes:

- SNALU62$_OK
- SNALU62$_ASSPEC
- SNALU62$_ASNEG

The SNALU62$ACTIVATE_SESSION procedure returns the following error codes:

- SNALU62$_PARERR
- SNALU62$_RESFNO
- SNALU62$_RESFRET

- SNALU62$_UNSUC

DIGITAL SNA Access Server for Windows NT Note

If the error occurred before the DIGITAL SNA Access Server for Windows NT attempted to connect with the IBM system, the error code is correct. If the error occurred while connecting to the IBM system, the DIGITAL SNA Access Server for Windows NT always returns SNALU62$_RESFRET and any sense code information is lost.

If an SNALU62$_RESFRET error code occurs while using the DIGITAL SNA Access Server for Windows NT, use the Microsoft Event Viewer to determine why the session activation failed. Refer to the Microsoft SNA Server and Event Viewer documentation for more information.

5.1.2 State Transition

When the return code is SNALU62$_ASSPEC or SNALU62$_ASNEG, the session is active.

5.2 SNALU62$DEACTIVATE_SESSION

SNALU62$DEACTIVATE_SESSION deactivates a session between the local logical unit and the remote logical unit. If the argument type indicates normal, then the session is not deactivated until all conversation ends. Do not use the synchronous version of this verb with type=SNALU62$K_DEACT_NORMAL unless there is no current conversation. If the argument type indicates cleanup, the session is deactivated immediately, regardless of the current state of the conversation or session. Any outstanding asynchronous verb, other than SNALU62$POST_ON_RECEIPT, completes before SNALU62$DEACTIVATE_SESSION completes.

Format:

status.wlc.v=SNALU62$DEACTIVATE_SESSION *(session_id.rlu.r, status_blk.wx.dx, [type.rlu.r], [efn.rlu.r], [astadr.szem.r], [astprm.rlu.r])*

Arguments:

status	When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits, together, represent the severity of the error. Returned by function value.
session_id	The unique identifier assigned by the Interface to the session. This value is returned by a previous call to SNALU62$ACTIVATE_SESSION or SNALU62$GET_ATTRIBUTES. Passed by reference.
status_blk	A data structure to receive status information on completion of the request. Passed by descriptor.

type

A value indicating the type of deactivation to be performed. The parameter takes the following values:

- SNALU62$K_DEACT_NORMAL indicates a normal deactivation. The session is not deactivated until all conversation has ended.
- SNALU62$K_DEACT_CLEANUP indicates a cleanup deactivation. The session is deactivated immediately regardless of the current state of the conversation or session. SNALU62$K_DEACT_CLEANUP is the default value.

Passed by reference.

efn

Number of the event flag to be set when this procedure completes. Passed by reference.

astadr

AST service routine to be executed when the procedure completes. Passed by reference.

astprm

AST parameter to be passed to the AST service routine specified by the *astadr* parameter. Passed by reference.

5.2.1 Status Codes

The SNALU62$DEACTIVATE_SESSION procedure returns the following successful status codes:

- SNALU62$_OK

The SNALU62$DEACTIVATE_SESSION procedure returns the following error codes:

- SNALU62$_PARERR
- SNALU62$_STAERR

5.3 SNALU62$DEFINE_REMOTE

The SNALU62$DEFINE_REMOTE initializes local LU parameters that control the operation of the local LU in conjunction with the remote LU.

Format:

status.wlc.v=SNALU62$DEFINE_REMOTE (*status_blk.wx.dx*,
qualified_luname.rt.dx,
local_luname.rt.dx,
[*uninterpreted_luname.rt.dx*],
[*initiate_type.rlu.r*],
[*parallel_session_support.rlu.r*],
[*cnos_support.rlu.r*],
[*lu_lu_password.rx.dx*],
[*security_acceptance.rlu.r*],
[*gwynode.rt.dx*],
[*accname.rt.dx*],
[*pu.rt.dx*],
[*sesaddr.rlu.r*],
[*plu.rt.dx*],
[*logon.rt.dx*],
[*user_data.rx.dx*],
[*authorization_password.rt.dx*],
[*process_id.rlu.r*])

Arguments:

status — When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits, together, represent the severity of the error. Returned by function value.

status_blk — A longword status vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.

qualified_luname — The network name for the remote LU that is being defined. This parameter is an arbitrary ASCII string, and in the present implementation, it does not have to be the real network LU name. ASCII string passed by descriptor.

local_luname — The local synonym for the remote LU. This name is an arbitrary ASCII string. It is the name that is passed to the ACTIVATE_SESSION or ALLOCATE procedures. ASCII string passed by descriptor.

uninterpreted_luname — Reserved for future use. This parameter is ignored. Passed by descriptor.

initiate_type

A session-initiation type that the local LU is to use on INITIATE requests. This parameter takes the following values:

- SNALU62$K_INITIATE_ONLY specifies an active connect request (default). A subsequent SNALU62$ACTIVATE_SESSION or SNALU62$ALLOCATE call specifying an LUNAME with the attribute SNALU62$K_INITIATE_ONLY, will immediately activate a session with the remote LU.
- SNALU62$K_INITIATE_OR_QUEUE specifies a passive connect request. A subsequent SNALU62$ACTIVATE_SESSION or SNALU62$ALLOCATE call specifying an LUNAME with the attribute SNALU62$K_INITIATE_OR_QUEUE, will wait for the remote LU to send a BIND to activate the session. Reserves the LU on the gateway node.

Passed by reference.

parallel_session_support

Reserved for future use. This parameter is ignored. Passed by reference.

cnos_support

Reserved for future use. This parameter is ignored. Passed by reference.

lu_lu_password

Specifies the LU-to-LU password (as defined on the IBM system) to be used for session level LU-to-LU verification during session activation. This parameter is a string of 8 bytes. Passed by descriptor.

DIGITAL SNA Access Server for Windows NT Note: Because the Microsoft SNA Server activates the session, the DIGITAL SNA Access Server for Windows NT does not itself support LU-to-LU verification during session activation. Therefore, the interface ignores this parameter when using the DIGITAL SNA Access Server for Windows NT. Use the Microsoft SNA Server Manager to supply this parameter.

security_acceptance

This parameter is ignored. Passed by reference. Reserved for future use.

gwynode

An ASCII string that specifies the name of the SNA gateway node. Passed by descriptor. For information about how to use this parameter to pass a TCP/IP gateway node name or an LU6.2 Server name, see Section 3.2 and its subsections.

DIGITAL SNA Access Server for Windows NT Note:
If you choose to use the DECnet transport between the APPC interface and the DIGITAL SNA Access Server for Windows NT, you must have DIGITAL PATHWORKS software installed on the DIGITAL SNA Access Server for Windows NT system.

accname

A gateway access name used in setting up a session with the host. This parameter is an ASCII string one to eight characters in length. If this parameter is omitted, or the access name definition does not contain all the required fields, then access information must be supplied by means of other parameters (PU, SESADDR, PLU, LOGON, DATA). Passed by descriptor.

DIGITAL SNA Access Server for Windows NT Note:
Access names on the DIGITAL SNA Access Server for Windows NT are treated slightly differently than on the older DECnet SNA gateways. For information about the differences, see Appendix J.

pu

An ASCII string specifying the SNA gateway or OpenVMS SNA Physical Unit (PU) name or, DIGITAL SNA Domain Gateway or DIGITAL SNA Peer Server Logical Unit (LU) name. Passed by descriptor.

- When connecting to the OpenVMS SNA or the DECnet SNA Gateways, this parameter contains an ASCII string in the form *dev-n*. The ASCII string specifies the gateway or OpenVMS SNA physical unit name.
- When connecting to a DIGITAL SNA Domain Gateway Gateway or DIGITAL SNA Peer Server, this parameter contains an ASCII string specifying up to an 8 character logical unit name.

DIGITAL SNA Access Server for Windows NT Note:
The DIGITAL SNA Access Server for Windows NT supports two uses of the *pu* and *sesaddr* parameters:

- Use the two parameters to specify an old-style *pu*[.*sesaddr*] LU name.
- Use the *pu* parameter to specify an LU name.

sesaddr	The gateway secondary logical unit (SLU) address for a DECnet SNA Gateway. This parameter is an integer in the range 1 to 255. This parameter is not used with DIGITAL SNA Domain Gateways or DIGITAL SNA Peer Server. Passed by reference. **DIGITAL SNA Access Server for Windows NT Note:** See note under *pu* parameter.
plu	A VTAM name for the primary logical unit (PLU). This parameter is an ASCII string one to eight characters in length. Passed by descriptor. **DIGITAL SNA Access Server for Windows NT Note:** This parameter must reference a remote LU alias defined on the DIGITAL SNA Access Server for Windows NT.
logon	A logon mode name. This parameter is an ASCII string one to eight characters in length. Passed by descriptor. **DIGITAL SNA Access Server for Windows NT Note:** This parameter must reference an APPC mode name defined on the Microsoft SNA Server.
user_data	A string of 1 to 128 characters that can be used to complete the logon to the PLU. This character string is not translated. Passed by descriptor. **DIGITAL SNA Access Server for Windows NT Note:** The interface ignores this parameter when using the DIGITAL SNA Access Server for Windows NT.
authorization_password	A password verified by the SNA gateway. Passed by descriptor.
process_id	Process identifier (PID) of a process logged into the system (default: current process). Passed by reference.

5.3.1 Status Codes

The SNALU62$DEFINE_REMOTE procedure returns the following successful status code:

- SNALU62$_OK

The SNALU62$DEFINE_REMOTE procedure returns the following error codes:

- SNALU62$_PARERR
- SNALU62$_UNSUC

5.4 SNALU62$DEFINE_TP

The SNALU62$DEFINE_TP procedure specifies a valid transaction program name with which a remote transaction program can initiate a conversation. If a transaction program on the remote LU attempts to initiate a conversation with a transaction program name on the local LU that has not been defined using SNALU62$DEFINE_TP, then conversation initiation will fail.

Format:

status.wlc.v=SNALU62$DEFINE_TP *(status_blk.wx.dx,*
tp_name.rt.dx,
[tp_status.rlu.r],
[tp_conversation_type.rlu.r],
[tp_synchronization_level.rlu.r],
[security_required.rlu.r],
[security_access.rlu.r],
[security_user_id.rx.dx],
[security_profile.rx.dx],
[pip.rlu.r],
[data_mapping.rlu.r],
[fmh_data.rlu.r],
[tp_privilege.rlu.r],
attach_routine.szem.r,
[attach_param.rlu.r])

Arguments:

status — When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits, together, represent the severity of the error. Returned by function value.

status_blk — A longword status vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.

tp_name	A valid transaction program name for this LU. If an attach request specifying this TPN arrives on a session from the remote LU, the ATTACH routine will be called. If the TPN in the ATTACH does not match any of the TPNs defined with this procedure, the ATTACH will be rejected and the ATTACH routine will not be called. The data supplied in this parameter is not translated. Passed by descriptor. **DIGITAL SNA Access Server for Windows NT Note:** All transaction program names must use characters contained in the standard translation tables supplied with the Microsoft SNA Server. If you use a user-defined translation table, you must choose characters for the TP name that are translated in the same manner as the standard translation tables supplied with the Microsoft SNA Server
tp_status	Reserved for future use. This parameter is ignored. Passed by reference.
tp_conversation_type	Reserved for future use. This parameter is ignored. Passed by reference.
tp_synchronization_level	Reserved for future use. This parameter is ignored. Passed by reference.
security_required	Reserved for future use. This parameter is ignored. Passed by reference.
security_access	Reserved for future use. This parameter is ignored. Passed by reference.
security_user_id	Reserved for future use. This parameter is ignored. Passed by descriptor.
security_profile	Reserved for future use. This parameter is ignored. Passed by descriptor.
pip	Reserved for future use. This parameter is ignored. Passed by reference.
data_mapping	Reserved for future use. This parameter is ignored. Passed by reference.
fmh_data	Reserved for future use. This parameter is ignored. Passed by reference.
tp_privilege	Reserved for future use. This parameter is ignored. Passed by reference.
attach_routine	A routine address to call if an ATTACH FMH arrives for the TPN. This parameter is required, even if you are changing only one of the other parameters. Passed by reference.

attach_param	A parameter to pass to the *attach_routine*. Passed by reference. This user-written attach routine has the following calling format: *attach_routine (attach_param.rlu.r,resource_id.rlu.r)*
where	
attach_param	is a parameter to pass to the *attach_routine*. Passed by reference.
resource_id	is the resource_id of the conversation that has now been attached. Further conversation verbs can now be issued on this conversation using this *resource_id*. Passed by reference.

5.4.1 Status Codes

The SNALU62$DEFINE_TP procedure returns the following successful status code:

- SNALU62$_OK

The SNALU62$DEFINE_TP procedure returns the following error codes:

- SNALU62$_PARERR
- SNALU62$_UNSUC
- SNALU62$_STAERR

5.5 SNALU62$DELETE

If a remote LU name is specified, the SNALU62$DELETE procedure deletes the specified remote LU.

Any sessions without active conversations are immediately deactivated. Sessions with active conversations will be deactivated when the conversation is deallocated. The verb completes when all sessions have been deactivated and all LU resources have been returned.

If a TP name is specified, the TPN is deleted from the list of valid TP names.

DIGITAL SNA Access Server for Windows NT Note

When using the DIGITAL SNA Access Server for Windows NT, this procedure only deletes information local to the APPC interface. It does not delete remote LU or TP name definitions on the Microsoft SNA Server.

Format:

status.wlc.v=SNALU62$DELETE *(status_blk.wx.dx, [local_lu.rx.dx], [remote_lu.rx.dx], [mode_name.rx.dx], [tp_name.rx.dx], [efn.rlu.r], [astadr.szem.r], [astprm.rlu.r])*

Arguments:

status	When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits, together, represent the severity of the error. Returned by function value.
status_blk	A longword status vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.
local_lu	Reserved for future use. This parameter is ignored. Passed by descriptor.
remote_lu	A remote LU name to be deleted from the list of valid remote LU names. Passed by descriptor.

mode_name	Reserved for future use. This parameter is ignored. Passed by descriptor.
tp_name	A valid TPN to be deleted for this LU. Passed by descriptor.
efn	An event flag to set when notification becomes due. If this parameter is omitted, event flag 0 is set. Passed by reference.
astadr	The address of an AST routine to call when notification becomes due. Passed by reference.
astprm	A parameter to pass to the AST routine when notification occurs. This AST routine is called with this parameter only. Passed by reference.

5.5.1 Status Codes

The SNALU62$DELETE procedure returns the following successful status code:

- SNALU62$_OK

The SNALU62$DELETE procedure returns the following error codes:

- SNALU62$_PARERR
- SNALU62$_UNSUC
- SNALU62$_STAERR

6

Compiling and Linking a Transaction Program

6.1 Creating and Compiling Your Program

Using the editor of your choice, create a source file containing the language source statements from one of the supported languages (Appendix B contains a programming example for most of the supported languages.)

Invoke the language required compiler to process the source statements. For example, enter the following command to compile a VAX BASIC program.

```
$ BASIC USERPROG2 Return
```

Verify that there are no syntax errors or violations of the language rules. The compiler will search any libraries you have specified, as well as any default libraries, to locate INCLUDE files referenced in the source program. Your program should contain an INCLUDE statement with the following reference:

```
INCLUDE 'SYS$LIBRARY:SNALU62DF.*'
```

where "*" is replaced with the appropriate extension for the language. See the programming example for the language being used.

If you are using MACRO language, assemble your MACRO program with the following DCL command:

```
$ MACRO/OBJECT=MYDIR:MYPROG SYS$LIBRARY:SNALU62DF+MYDIR:MYPROG Return
```

where MYDIR and MYPROG are your directory and program.

If there are no errors, the compiler creates an object module. If errors are reported, determine the line(s) containing the errors, edit the program to correct the errors, and then recompile the program.

6.2 Linking Your Program to the Shareable Program Image

After you have compiled the source statements, you are ready to link them with the shareable image of the APPC/LU6.2 Programming Interface procedures. Your image shares these procedures with other images (on the condition that the shareable image is installed with the /SHAREABLE attribute with the VMSINSTAL utility). For additional information, see the *OpenVMS Install Utility Manual*.

To use the shareable interface procedures, you must specify a linker options file. This links your executable image with the shareable image, SYS$SHARE:SNALU62SH.EXE. For example:

```
$ LINK/MAP USERPROG2, SYS$INPUT:/OPTION  Return
SYS$SHARE:SNALU62SH/SHARE  Return
Ctrl/Z
$
```

The following example links your executable image with the debugger and the shareable image SYS$SHARE:SNALU62SH.EXE. You must specify a linker options file.

```
$ LINK/MAP/DEBUG USERPROG2, SYS$INPUT:/OPTION  Return
SYS$SHARE:SNALU62SH/SHARE  Return
Ctrl/Z
$
```

Once you have compiled and linked your program, you are ready to run it. For a detailed description of the LINK command and any additional options, see the *OpenVMS Linker Utility Manual*. Also, refer to your language manual for additional information.

A

Summary of Parameter Notation

Table A-1 shows the parameter notation for the DIGITAL SNA APPC/LU6.2 Programming Interface for OpenVMS. For further information about notations and their meanings see *Introduction to OpenVMS System Routines*.

Position the parameter as follows:

<name>.<access type><data type>.<passing mech> <parameter form>

where

1. <name> is a mnemonic for the parameter.
2. <access type> is a single letter denoting the type of access that the procedure will (or can) make to the argument.
3. <data type> is a letter denoting the primary data type with trailing qualifier letters to further identify the data type. The routine must reference only the size specified to avoid improper access violations.
4. <passing mechanism> is a single letter indicating the parameter passing mechanism that the called routine expects.
5. <parameter form> is a letter denoting the form of the argument.

Table A–1 Parameter Notation

Name	Meaning
	<access type>
c	Call after stack unwind
f	Function call (before return)
j	JMP after unwind
m	Modify access

(continued on next page)

Table A–1 (Cont.) Parameter Notation

Name	Meaning
	<access type>
r	Read-only access
s	Call without stack unwinding
w	Write-only access
	<data type>
a	Virtual address
ad	Absolute data and time
arb	8-bit relative virtual address
arl	32-bit relative virtual address
arw	16-bit relative virtual address
b	Byte integer (signed)
blv	Bound label value
bpv	Bound procedure value
bu	Byte logical (unsigned)
c	Single character
cit	COBOL intermediate temporary
cp	Character pointer
d	D_floating
dc	D_floating complex
dsc	Descriptor (used by descriptors)
f	F_floating
fc	F_floating complex
g	G_floating
gc	G_floating complex

(continued on next page)

Table A–1 (Cont.) Parameter Notation

Name	Meaning
	<data type>
h	H_floating
hc	H_floating complex
l	Longword integer (signed)
lc	Longword return status
lu	Longword logical (unsigned)
nl	Numeric string, left separate sign
nlo	Numeric string, left overpunched sign
nr	Numeric string, right separate sign
nro	Numeric string, right overpunched sign
nu	Numeric string, unsigned
nz	Numeric string, zoned sign
o	Octaword integer (signed)
ou	Octaword logical (unsigned)
p	Packed decimal string
q	Quadword integer (signed)
qu	Quadword logical (unsigned)
r	Record
t	Character-coded text string
u	Smallest addressable storage unit
v	Aligned bit string
vt	Varying character-coded test string
vu	Unaligned bit string
w	Word integer (signed)
wu	Word logical (unsigned)

(continued on next page)

Table A–1 (Cont.) Parameter Notation

Name	Meaning
	<data type>
x	Data type in descriptor
z	Unspecified
zem	Procedure entry mask
zi	Sequence of instruction
	<passing mechanism>
d	By descriptor
r	By reference
v	By immediate value
	<parameter form>
_	Scalar
a	Array reference or descriptor
d	Dynamic string descriptor
nca	Noncontiguous array descriptor
p	Procedure reference or descriptor
s	Fixed-length string descriptor
sd	Scalar decimal descriptor
uba	Unaligned bit string array descriptor
ubs	Unaligned bit string descriptor
vs	Varying string descriptor
vsa	Varying string array descriptor
x	Class type in descriptor
x1	Fixed-length or dynamic string descriptor

B

Programming Examples

The following programming examples show you how to make calls to the DIGITAL SNA APPC/LU6.2 Programming Interface for OpenVMS from your OpenVMS transaction programs. These examples include comments to help solve problems you may have with the different languages.

These programs have been proven in field test situations to be valid for use with DIGITAL SNA products. They may not, however, be valid at all installations; use the examples as a guide for developing your transaction programs. The examples use the following languages:

- FORTRAN
- Pascal
- VAX BASIC
- MACRO
- COBOL
- VAX PL/I
- C

Comments follow each programming example.

Allocate event flags before use. Use the LIB$GET_EF (or LIB$RESERVE_EF) and LIB$FREE_EF Run Time Library routines to handle allocation and deallocation.

With the limited number of event flags, it is possible to use event flags that have not been specifically allocated and resulting in multiple asynchronous operations sharing these limited resources. Problems can occur when one asynchronous operation sets a shared event flag which can indicate to other users of the event flag that their operation has completed.

If there are not enough event flags for the number of conversations, use the $SYNC routine and specify the event flag and cleared (unique) status_vector (as iosb). Use a different event flag/status_vector combination for each conversion. $SYNC does not complete until **both** the event flag is set **and** the status_vector is filled in. See the *OpenVMS System Services Reference Manual* and the *OpenVMS Run-Time Library Routines Reference Manual* for further information.

B.1 FORTRAN Programming Example

This FORTRAN program connects to AIBR, a sample transaction running under CICS. After establishing a connect with AIBR, the OpenVMS transaction program prompts you for the record number you want to send. This record number is converted to EBCDIC and sent to AIBR. The program receives data from the AIBR transaction, converts the data to ASCII format, and displays at your terminal.

```
        PROGRAM LU62_EXAMPLE
C ********************************************************************
C *                                                                  *
C * This program allocates a session with the AIBR transaction      *
C * running under CICS 1.7.  It prompts the user for a record       *
C * number, requests the record and displays it.  A deallocate      *
C * message is sent and the session is terminated.  Proper operation *
C * can be verified with SNATRACE.  This program is intended to     *
C * represent the minimal framework for an SNA Gateway Application   *
C * Interface program.                                               *
C *                                                                  *
C ********************************************************************

        IMPLICIT NONE

C
C Declaration
C
        INCLUDE 'SYS$LIBRARY:SNALU62DF'
 INTEGER*4 LIB$GET_EF
        INTEGER*4 LIB$GET_INPUT, LIB$TRA_ASC_EBC, LIB$TRA_EBC_ASC
        INTEGER*4 LIB$PUT_OUTPUT
        INTEGER*4 SYS$CLREF, SYS$WAITFR, SYS$PUTMSG
```

```
        INTEGER*4 RETURN_CODE,RESOURCE_ID,I_STAT, EVENT_FLAG
        INTEGER*4 RECORD_NUMBER_LENGTH, DSC$K_DTYPE_T, DSC$K_CLASS_D
        INTEGER*4 RTS_REC,RECV_DATA_LENGTH,WHAT_RECEIVED
        INTEGER*4 STATUS_VECTOR(SNALU62$K_MIN_STATUS_VECTOR)
        INTEGER*2 INPUT_SIZE
        CHARACTER*6 NODE_NAME/'ALACK'/
        CHARACTER*8 ACC_NAME/'CICSLU62'/
        CHARACTER*6 LUNAME/'DUMMY'/,RECORD_NUMBER
        CHARACTER*4 ASCII_TPN_NAME/'AIBR'/,TPN_NAME

        PARAMETER (RECORD_NUMBER_LENGTH = 6,
     1             DSC$K_DTYPE_T = 14,
     2             DSC$K_CLASS_D = 2)

        STRUCTURE /DSC$DESCRIPTOR/
            INTEGER*2   DSC$W_LENGTH
            BYTE        DSC$B_DTYPE
            BYTE        DSC$B_CLASS
            INTEGER*4   DSC$A_POINTER
        END STRUCTURE    !DSC$DESCRIPTOR
        RECORD /DSC$DESCRIPTOR/ RECV_DATA
        RECV_DATA.DSC$B_DTYPE = DSC$K_DTYPE_T
        RECV_DATA.DSC$B_CLASS = DSC$K_CLASS_D
        RECV_DATA.DSC$W_LENGTH = 0
        RECV_DATA.DSC$A_POINTER = 0

C
C Allocate event flag
C

        I_STAT = LIB$GET_EF(%REF(EVENT_FLAG))
        IF (.NOT.I_STAT) CALL LIB$STOP(%VAL(I_STAT))

C
C Define LU name
C
        RETURN_CODE = SNALU62$DEFINE_REMOTE(%DESCR(STATUS_VECTOR),
     1    LUNAME,LUNAME,
     2    ,,,,,,
     3    NODE_NAME,ACC_NAME)
        IF (.NOT.RETURN_CODE) THEN
            I_STAT = SYS$PUTMSG(STATUS_VECTOR)
            STOP 'DEFINE_REMOTE verb failed'
        END IF

C
C Translate transaction program name to EBCDIC and allocate
C conversation
C
        I_STAT = LIB$TRA_ASC_EBC(ASCII_TPN_NAME, TPN_NAME)
        IF (.NOT.I_STAT) CALL LIB$STOP(%VAL(I_STAT))

        I_STAT = SYS$CLREF(%VAL(EVENT_FLAG))
        IF (.NOT.I_STAT) CALL LIB$STOP(%VAL(I_STAT))
```

```
        RETURN_CODE = SNALU62$ALLOCATE(RESOURCE_ID,
     1    %DESCR(STATUS_VECTOR),%REF(SNALU62$K_OTHER),
     2    LUNAME,,%DESCR(TPN_NAME),
     3    %REF(SNALU62$K_MAPPED_CONVERSATION),
     4    %REF(SNALU62$K_WHEN_SESSION_ALLOC),
     5    %REF(SNALU62$K_SL_CONFIRM),
     6    ,,,,,EVENT_FLAG)

        IF (.NOT.RETURN_CODE) THEN
            I_STAT = SYS$PUTMSG(STATUS_VECTOR)
            STOP 'ALLOCATE verb failed'
        END IF

        I_STAT = SYS$WAITFR(%VAL(EVENT_FLAG))
        IF (.NOT.I_STAT) CALL LIB$STOP(%VAL(I_STAT))

C
C Ask for record number, translate to EBCDIC and send data
C
        I_STAT = LIB$GET_INPUT(RECORD_NUMBER,
     1    'Enter record number: ', INPUT_SIZE)
        IF (.NOT.I_STAT) CALL LIB$STOP(%VAL(I_STAT))

        IF (INPUT_SIZE .EQ. 0) GOTO 999
        I_STAT = LIB$TRA_ASC_EBC(RECORD_NUMBER, RECORD_NUMBER)
        IF (.NOT.I_STAT) CALL LIB$STOP(%VAL(I_STAT))

        RETURN_CODE = SNALU62$SEND_DATA(RESOURCE_ID,
     1    %DESCR(STATUS_VECTOR), RECORD_NUMBER,
     2    %REF(RECORD_NUMBER_LENGTH),
     3    RTS_REC)

        IF (.NOT.RETURN_CODE) THEN
            I_STAT = SYS$PUTMSG(STATUS_VECTOR)
            STOP 'SEND_DATA verb failed'
        END IF

C
C Receive record from CICS/AIBR
C
        RECV_DATA_LENGTH = 0
        RETURN_CODE = SNALU62$RECEIVE_AND_WAIT(RESOURCE_ID,
     1    %DESCR(STATUS_VECTOR),0,RECV_DATA_LENGTH,RTS_REC,
     2    RECV_DATA,WHAT_RECEIVED)

        IF (.NOT.RETURN_CODE) THEN
            I_STAT = SYS$PUTMSG(STATUS_VECTOR)
            STOP 'RECEIVE_AND_WAIT verb failed'
        END IF

C
C Convert record to ASCII and display on user terminal
C
        I_STAT = LIB$TRA_EBC_ASC(RECV_DATA, RECV_DATA)
        IF (.NOT.I_STAT) CALL LIB$STOP(%VAL(I_STAT))
```

```
        I_STAT = LIB$PUT_OUTPUT(RECV_DATA)
        IF (.NOT.I_STAT) CALL LIB$STOP(%VAL(I_STAT))
C
C Receive DEALLOCATE
C
        RECV_DATA_LENGTH = 0
        RETURN_CODE = SNALU62$RECEIVE_AND_WAIT(RESOURCE_ID,
     1    %DESCR(STATUS_VECTOR),0,RECV_DATA_LENGTH,RTS_REC,
     2    RECV_DATA,WHAT_RECEIVED)
        IF (RETURN_CODE .NE. SNALU62$_DEALNOR) THEN
            I_STAT = SYS$PUTMSG(STATUS_VECTOR)
            STOP 'RECEIVE_AND_WAIT verb failed'
        END IF
C
C Deallocate conversation
C
        RETURN_CODE = SNALU62$DEALLOCATE(RESOURCE_ID,
     1    %DESCR(STATUS_VECTOR), %REF(SNALU62$K_LOCAL))
C
C Delete LU name
C
        RETURN_CODE = SNALU62$DELETE(%DESCR(STATUS_VECTOR),,
     1    LUNAME)
 999    I_STAT = SYS$PUTMSG(STATUS_VECTOR)
        END
```

COMMENTS

1. Include the LU6.2 library.
2. A real transaction program would check the *what_received* parameter and take appropriate action. In this example, a fixed-sequence message exchange is assumed.
3. You can use OpenVMS library routines to do parts of your application, such as translating ASCII to EBCDIC or vice versa.
4. The AIBR transaction deallocates the conversation after the record has been transmitted. You must issue the SNALU62$RECEIVE_AND_WAIT procedure to receive notification and set the conversation state. Conversations can then be locally deallocated and reused.
5. Deactivate the session in a clean manner. Use the SNALU62$DELETE procedure to delete the remote LU defined with SNALU62$DEFINE_REMOTE.

B.2 Pascal Programming Example

This Pascal program connects to AIBR, a sample transaction running under CICS. After establishing a connect with AIBR, the OpenVMS transaction program prompts you for the record number you want to send. This record number is converted to EBCDIC and sent to AIBR. The program receives data from the AIBR transaction, converts the data to ASCII format, and displays at your terminal.

```
[INHERIT  ('SYS$LIBRARY:SNALU62DF.PEN', 'SYS$LIBRARY:STARLET.PEN')]
PROGRAM   LU62_EXAMPLE (INPUT,OUTPUT);
(*****************************************************************)
(*                                                               *)
(* This program allocates a session with the AIBR transaction    *)
(* running under CICS 1.7.  It prompts the user for a record     *)
(* number, requests the record and displays it.  A deallocate    *)
(* message is sent and the session is terminated.  Proper operation *)
(* can be verified with SNATRACE.  This program is intended to   *)
(* represent the minimal framework for an SNA Gateway Application *)
(* Interface program.                                            *)
(*                                                               *)
(*****************************************************************)

[HIDDEN] TYPE   (**** Pre-declared data types ****)
        $BYTE = [BYTE] -128..127;
        $WORD = [WORD] -32768..32767;
        $UBYTE = [BYTE] 0..255;
        $UWORD = [WORD] 0..65535;

[ASYNCHRONOUS] FUNCTION LIB$GET_EF(
                   VAR EVENT_FLAG: [VOLATILE] UNSIGNED)
                               : INTEGER; EXTERNAL;

[EXTERNAL] FUNCTION LIB$GET_INPUT
                        (VAR IN_BUFFER : [CLASS_S] PACKED ARRAY
                                [$l1..$u1 : INTEGER] OF CHAR;
                         PROMPT : [CLASS_S] PACKED ARRAY
                                [$l2..$u2 : INTEGER] OF CHAR;
                         VAR LENGTH : $WORD)
                           : INTEGER; EXTERNAL;

[EXTERNAL] FUNCTION LIB$STOP(
                STATUS : [IMMEDIATE,UNSAFE] UNSIGNED)
                           : INTEGER; EXTERNAL;
```

```
[EXTERNAL] FUNCTION LIB$TRA_ASC_EBC
                        (VAR IN_BUFFER : [CLASS_S] PACKED ARRAY
                                [$l1..$u1:INTEGER] OF CHAR;
                         VAR OUT_BUFFER : [CLASS_S] PACKED ARRAY
                                [$l2..$u2:INTEGER] OF CHAR)
                          : INTEGER; EXTERNAL;

[EXTERNAL] FUNCTION LIB$TRA_EBC_ASC
                        (VAR IN_BUFFER : [CLASS_S] PACKED ARRAY
                                [$l1..$u1:INTEGER] OF CHAR;
                         VAR OUT_BUFFER : [CLASS_S] PACKED ARRAY
                                [$l2..$u2:INTEGER] OF CHAR)
                          : INTEGER; EXTERNAL;

(*****************************************************************)
(*    Declaration                                                *)
(*****************************************************************)

LABEL 999;

CONST
    BUFFER_LENGTH = 2000;
    RECORD_NUMBER_LENGTH = 6;

VAR
    I_STATUS : UNSIGNED;
    SNA_STATUS : UNSIGNED;
    LIB_STATUS : UNSIGNED;
    STATUS_VECTOR : PACKED ARRAY
                    [1..SNALU62$K_MIN_STATUS_VECTOR]
                    OF CHAR;
    EVENT_FLAG : UNSIGNED;
    EVENT_CODE : INTEGER;
    RESOURCE_ID : INTEGER;
    RTS_REC : INTEGER;
    RECV_DATA_LENGTH : $WORD;
    WHAT_RECEIVED : INTEGER;
    INPUT_SIZE : $WORD;
    NODE_NAME : PACKED ARRAY [1..8] OF CHAR;
    ACCESS_NAME : PACKED ARRAY [1..8] OF CHAR;
    RECV_DATA : PACKED ARRAY [1..BUFFER_LENGTH] OF CHAR;
    LUNAME : PACKED ARRAY [1..5] OF CHAR;
    RECORD_NUMBER : PACKED ARRAY [1..RECORD_NUMBER_LENGTH] OF CHAR;
    ASCII_TPN_NAME : PACKED ARRAY [1..4] OF CHAR;
    TPN_NAME : PACKED ARRAY [1..4] OF CHAR;
    RN_PROMPT : PACKED ARRAY [1..22] OF CHAR;
    I : INTEGER;
```

```
BEGIN
    ASCII_TPN_NAME := 'AIBR';
    LUNAME := 'DUMMY';
    NODE_NAME := 'ALACK';
    ACCESS_NAME := 'CICSLU62';
    RN_PROMPT := 'Enter record number: ';
    RTS_REC := 0;
    EVENT_FLAG := 0;
(*****************************************************************)
(*  DEFINE LU NAME                                               *)
(*****************************************************************)
    SNA_STATUS := SNALU62$DEFINE_REMOTE(STATUS_VECTOR,
                                        LUNAME,
                                        LUNAME,,,,,,,
                                        NODE_NAME,
                                        ACCESS_NAME);

    IF (NOT SNA_STATUS :: BOOLEAN)
    THEN
        BEGIN
        WRITELN ('DEFINE failed');
        LIB_STATUS := $PUTMSG(STATUS_VECTOR);
        GOTO 999
        END;

(*****************************************************************)
(* Translate transaction program name to EBCDIC, allocate event  *)
(* flag, and allocate conversation                               *)
(*****************************************************************)
    LIB_STATUS := LIB$TRA_ASC_EBC (ASCII_TPN_NAME, TPN_NAME);

    LIB_STATUS := LIB$GET_EF( EVENT_FLAG );
    IF (NOT LIB_STATUS :: BOOLEAN)
    THEN
        LIB_STATUS := LIB$STOP(LIB_STATUS);

    I_STATUS := $CLREF( EVENT_FLAG );

    SNA_STATUS := SNALU62$ALLOCATE(RESOURCE_ID,
                                   STATUS_VECTOR,
                                   %REF(SNALU62$K_OTHER),
                                   LUNAME,,
                                   TPN_NAME,
                                   %REF(SNALU62$K_MAPPED_CONVERSATION),
                                   %REF(SNALU62$K_WHEN_SESSION_ALLOC),
                                   %REF(SNALU62$K_SL_CONFIRM),,,,,,
                                   %REF(EVENT_FLAG));

    I_STATUS := $WAITFR( EVENT_FLAG );
```

```
    IF (NOT SNA_STATUS :: BOOLEAN)
    THEN
        BEGIN
        WRITELN ('ALLOCATE Failed');
        LIB_STATUS := $PUTMSG(STATUS_VECTOR);
        GOTO 999
        END;

(*********************************************************************)
(*      Ask for record number, translate to EBCDIC and send data     *)
(*********************************************************************)

    LIB_STATUS := LIB$GET_INPUT(RECORD_NUMBER,
                                RN_PROMPT,
                                INPUT_SIZE);
    IF (NOT LIB_STATUS :: BOOLEAN)
    THEN
        LIB_STATUS := LIB$STOP(LIB_STATUS);

    IF (INPUT_SIZE = 0) THEN GOTO 999;

    LIB_STATUS := LIB$TRA_ASC_EBC (RECORD_NUMBER, RECORD_NUMBER);

    IF (NOT LIB_STATUS :: BOOLEAN)
    THEN
        LIB_STATUS := LIB$STOP(LIB_STATUS);

    SNA_STATUS := SNALU62$SEND_DATA(RESOURCE_ID,
                                    STATUS_VECTOR,
                                    RECORD_NUMBER,
                                    %REF(RECORD_NUMBER_LENGTH),,,
                                    RTS_REC);

    IF (NOT SNA_STATUS :: BOOLEAN)
    THEN
        BEGIN
        WRITELN ('SEND_DATA failed');
        LIB_STATUS := $PUTMSG(STATUS_VECTOR);
        GOTO 999
        END;

(*********************************************************************)
(*      Receive record form CICS/AIBR                                *)
(*********************************************************************)

    RECV_DATA_LENGTH := BUFFER_LENGTH;
    SNA_STATUS := SNALU62$RECEIVE_AND_WAIT(RESOURCE_ID,
                                           STATUS_VECTOR,
                                           0,
                                           RECV_DATA_LENGTH,
                                           RTS_REC,
                                           RECV_DATA,
                                           WHAT_RECEIVED);
```

```
    IF (NOT SNA_STATUS :: BOOLEAN)
    THEN
        BEGIN
        LIB_STATUS := $PUTMSG(STATUS_VECTOR);
        GOTO 999
        END;

(*********************************************************************)
(*      Convert record to ASCII and display on user terminal         *)
(*********************************************************************)

    LIB_STATUS := LIB$TRA_EBC_ASC(RECV_DATA,
                                  RECV_DATA);

    FOR I := 1 TO RECV_DATA_LENGTH DO
        IF ROUND(I/80) = I/80
        THEN
            WRITELN(RECV_DATA[I])
        ELSE
            WRITE(RECV_DATA[I]);
    WRITELN;

(*********************************************************************)
(*      Receive DEALLOCATE                                           *)
(*********************************************************************)

    RECV_DATA_LENGTH := BUFFER_LENGTH;
    SNA_STATUS := SNALU62$RECEIVE_AND_WAIT(RESOURCE_ID,
                                           STATUS_VECTOR,
                                           0,
                                           RECV_DATA_LENGTH,
                                           RTS_REC,
                                           RECV_DATA,
                                           WHAT_RECEIVED);

    IF (SNA_STATUS <> SNALU62$_DEALNOR)
    THEN
        BEGIN
            WRITELN ('Failed to receive deallocation');
            WRITELN ('RECEIVE failed');
            LIB_STATUS := $PUTMSG(STATUS_VECTOR);
            GOTO 999
        END;

(*********************************************************************)
(*      Deallocate conversation                                      *)
(*********************************************************************)

    SNA_STATUS := SNALU62$DEALLOCATE(RESOURCE_ID,
                                     STATUS_VECTOR,
                                     %REF(SNALU62$K_LOCAL));
```

```
    IF (NOT SNA_STATUS :: BOOLEAN)
    THEN
         BEGIN
             WRITELN ('DEALLOCATE failed');
             LIB_STATUS := $PUTMSG(STATUS_VECTOR);
             GOTO 999
         END;
(*********************************************************************)
(*      Delete LU name                                               *)
(*********************************************************************)
    SNA_STATUS := SNALU62$DELETE(STATUS_VECTOR,,
                                 LUNAME);
999:    LIB_STATUS := $PUTMSG(STATUS_VECTOR);
        END.
```

COMMENTS

1. Specify the LU6.2 environment file.
2. If you are going to send more data, the program should check the *rts_rec* parameter between calls.
3. You can use OpenVMS library routines to do parts of your application, such as translating ASCII to EBCDIC or vice versa.
4. The AIBR transaction deallocates the conversation after the record has been transmitted. You must issue the SNALU62$RECEIVE_AND_WAIT procedure to receive notification and set the conversation state. Conversations can then be locally deallocated and reused.
5. Deactivate the session in a clean manner. Use the SNALU62$DELETE procedure to delete the remote LU defined with SNALU62$DEFINE_REMOTE.

B.3 Pascal Symbol and Structure Definitions

The APPC/LU6.2 Programming Interface supplies symbol and structure definitions used by the application program. For Pascal, these definitions are provided in a source code file (SNALU62DF.PAS), and an environment file (SNALU62DF.PEN). The environment file was generated with PASCAL V4.0-02 on OpenVMS VAX. If you do not have this version of Pascal, you may need to generate a new environment file by editing the SNALU62DF.PAS file located in the SYS$LIBRARY area and removing the comment delimiters: "(*" and "*)" surrounding the two statements MODULE SNALU62DF and .END.

Exit the file and enter the following command:

```
$ PASCAL/NOOBJECT/ENVIRONMENT=SNALU62DF.PEN SYS$LIBRARY:SNALU62DF.PAS
```

Please remember to go back and delete the edited file of SNALU62DF.PAS which has the comment delimiters removed.

B.4 BASIC Programming Example

This BASIC program connects to AIBR, a sample transaction running under CICS. After establishing a connect with AIBR, the OpenVMS transaction program prompts you for the record number you want to send. This record number is converted to EBCDIC and sent to AIBR. The program receives data from the AIBR transaction, converts the data to ASCII format, and displays at your terminal.

```
1       %TITLE  "LU62 BASIC PROGRAM EXAMPLE"
!*********************************************************************
!*                                                                   *
!* This program allocates a session with the AIBR transaction        *
!* running under CICS 1.7.  It prompts the user for a record         *
!* number, requests the record and displays it.  A deallocate        *
!* message is sent and the session is terminated.  Proper operation  *
!* can be verified with SNATRACE.  This program is intended to       *
!* represent the minimal framework for an SNA Gateway Application    *
!* Interface program.                                                *
!*                                                                   *
!*********************************************************************
7       OPTION SIZE = INTEGER LONG      ! all integer vars are 4 bytes
                                        ! long

        %INCLUDE "SYS$LIBRARY:SNALU62DF"

9               DECLARE                                             &
                 LONG CONSTANT                                      &
                        LU62$K_OTHER = X'00000002'L,                &
                        LU62$K_MAPPED_CONVERSATION = X'00000004'L,  &
                        LU62$K_WHEN_SESSION_ALLOCATED = X'00000005'L,&
                        LU62$K_SL_CONFIRM = X'00000009'L,           &
                        LU62$K_LOCAL = X'00000020'L,                &
                        LU62$K_ABEND_PROG = X'0000001D'L,           &
                        LU62$K_CONFIRM_DEALLOCATE = X'0000002E'L

15      EXTERNAL INTEGER FUNCTION   LIB$TRA_ASC_EBC,SYS$PUTMSG,     &
                                    LIB$TRA_EBC_ASC,LIB$GET_EF,      &
                                    SYS$CLREF,SYS$WAITFR
```

```
30      !
        ! Declaration
        !
        DECLARE                                                         &
                INTEGER                                                 &
                        STATUS_VECTOR(64),RETURN_CODE,ISTAT,            &
                        RESOURCE_ID,RTS_REC,RECV_DATA_LENGTH,           &
                        WHAT_RECEIVED,RECV_BUFFER_LENGTH,EVENT_FLAG,    &
                STRING                                                  &
                        TPN_NAME,GWY_NAME,RECV_DATA,RECORD_NUMBER,      &
                        ACCESS_NAME,ASCII_TPN_NAME,LU_NAME

50      READ GWY_NAME,ACCESS_NAME,ASCII_TPN_NAME,LU_NAME,               &
                RECV_BUFFER_LENGTH
        DATA ALACK,CICSLU62,AIBR,DUMMY,2000

90      !
        ! Define LU name
        !
        RETURN_CODE = SNALU62$DEFINE_REMOTE (STATUS_VECTOR() BY DESC,&
                LU_NAME BY DESC,LU_NAME BY DESC,0,0,0,0,0,0,            &
                GWY_NAME BY DESC,ACCESS_NAME BY DESC)

        IF ((RETURN_CODE AND 1%) <> 1) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR() BY REF)
                PRINT "DEFINE verb failed"
                GO TO 999

110     !
        ! Translate transaction program name to EBCDIC, allocate an
 ! event flag, and allocate the conversation
        !
        ISTAT = LIB$TRA_ASC_EBC (ASCII_TPN_NAME BY DESC,                &
                TPN_NAME BY DESC)

        ISTAT = LIB$GET_EF (EVENT_FLAG BY REF)

        ISTAT = SYS$CLREF(EVENT_FLAG BY VALUE)

        RETURN_CODE = SNALU62$ALLOCATE (RESOURCE_ID BY REF,             &
                STATUS_VECTOR() BY DESC, LU62$K_OTHER BY REF,           &
                LU_NAME BY DESC,0,TPN_NAME BY DESC,                     &
                LU62$K_MAPPED_CONVERSATION BY REF,                      &
                LU62$K_WHEN_SESSION_ALLOCATED BY REF,                   &
                LU62$K_SL_CONFIRM BY REF,0,0,0,0,0,                     &
                EVENT_FLAG BY REF)

        ISTAT = SYS$WAITFR(EVENT_FLAG BY VALUE)

        IF ((RETURN_CODE AND 1%) <> 1) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR() BY REF)
                PRINT "ALLOCATE verb failed"
                GO TO 999
```

```
130     !
        ! Ask for record number, translate to EBCDIC and send data
        !
        INPUT "Enter record number: ";RECORD_NUMBER

        ISTAT = LIB$TRA_ASC_EBC (RECORD_NUMBER BY DESC,              &
                RECORD_NUMBER BY DESC)

150     RETURN_CODE = SNALU62$SEND_DATA (RESOURCE_ID BY REF,         &
                STATUS_VECTOR() BY DESC, RECORD_NUMBER BY DESC,      &
                0, RTS_REC BY REF)

        IF ((RETURN_CODE AND 1%) <> 1) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR() BY REF)
                PRINT "SEND_DATA verb failed"
                GO TO 999

170     !
        ! Receive record from CICS/AIBR
        !
        RECV_DATA_LENGTH = RECV_BUFFER_LENGTH
        RETURN_CODE = SNALU62$RECEIVE_AND_WAIT (RESOURCE_ID BY REF,  &
                STATUS_VECTOR() BY DESC,0,RECV_DATA_LENGTH BY REF,   &
                RTS_REC BY REF,RECV_DATA BY DESC,WHAT_RECEIVED BY REF)

        IF ((RETURN_CODE AND 1%) <> 1) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR() BY REF)
                PRINT "RECEIVE_AND_WAIT verb failed"
                GO TO 999

190     !
        ! Convert record to ASCII and display on user terminal
        !
        ISTAT = LIB$TRA_EBC_ASC (RECV_DATA BY DESC,RECV_DATA BY DESC)
        PRINT RECV_DATA

210     !
        ! Receive DEALLOCATE
        !
        RECV_DATA_LENGTH = RECV_BUFFER_LENGTH
        RETURN_CODE = SNALU62$RECEIVE_AND_WAIT (RESOURCE_ID BY REF, &
                STATUS_VECTOR() BY DESC,0,RECV_DATA_LENGTH BY REF,  &
                RTS_REC BY REF,RECV_DATA BY DESC,WHAT_RECEIVED BY REF)

        IF (RETURN_CODE <> SNALU62$_DEALNOR)
        THEN
                IF ((RETURN_CODE AND 1%) <> 1) THEN
                        ISTAT = SYS$PUTMSG(STATUS_VECTOR() BY REF)
                        PRINT "RECEIVE verb failed"
                        GO TO 999
                END IF
        END IF
```

```
230     !
        ! Deallocate conversation
        !
        RETURN_CODE = SNALU62$DEALLOCATE (RESOURCE_ID BY REF,       &
                STATUS_VECTOR() BY DESC,LU62$K_LOCAL BY REF)

        IF ((RETURN_CODE AND 1%) <> 1) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR() BY REF)
                PRINT "DEALLOCATE verb failed"
                GO TO 999
        END IF

        !
        ! Delete LU name
        !
        RETURN_CODE = SNALU62$DELETE (STATUS_VECTOR() BY DESC,      &
                                      0,LU_NAME BY DESC)

        IF ((RETURN_CODE AND 1%) <> 1) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR() BY REF)
                PRINT "DELETE verb failed"
        END IF

999     ISTAT = SYS$PUTMSG(STATUS_VECTOR() BY REF)

        END
```

COMMENTS

1. Include LU6.2 error codes.
2. You must define these symbols. Appendix F lists the symbols.
3. If you are going to send more data, the program should check the *rts_rec* parameter between calls.
4. A real transaction program would check the *what_received* parameter and take appropriate action (see Sections 2.2.1 and 4.11). In this example, a fixed-sequence message exchange is assumed.
5. You can use OpenVMS library routines to do parts of your application, such as translating ASCII to EBCDIC or vice versa.
6. The AIBR transaction deallocates the conversation after the record has been transmitted. You must issue the SNALU62$RECEIVE_AND_WAIT procedure to receive notification and set the conversation state. Conversations can then be locally deallocated and reused.
7. Deactivate the session in a clean manner. Use the SNALU62$DELETE procedure to delete the remote LU defined with SNALU62$DEFINE_REMOTE.

B.5 MACRO Programming Example

This MACRO program connects to AIBR, a sample transaction running under CICS. After establishing a connect with AIBR, the OpenVMS transaction program prompts you for the record number you want to send. This record number is converted to EBCDIC and sent to AIBR. The program receives data from the AIBR transaction, converts the data to ASCII format, and displays at your terminal.

```
         .TITLE TEST_LU62
;*********************************************************************
;*                                                                   *
;* This program allocates a session with the AIBR transaction       *
;* running under CICS 1.7.  It prompts the user for a record        *
;* number, requests the record and displays it.  A deallocate       *
;* message is sent and the session is terminated.  Proper operation *
;* can be verified with SNATRACE.  This program is intended to       *
;* represent the minimal framework for an SNA Gateway Application    *
;* Interface program.                                                *
;*                                                                   *
;*********************************************************************
         $DSCDEF
         SNALU62_DEFS
;
; Declaration
;

.PSECT RWDATA,WRT,NOEXE,QUAD
```

```
PROMPT : .ASCID  /Enter record number: /
STATUS : .BLKL   10
RESC_ID: .LONG   0                              ;session-id
STS_VEC: .BLKB   SNALU62$K_MIN_STATUS_VECTOR    ;status-vector
STS_DSC: .LONG   SNALU62$K_MIN_STATUS_VECTOR
         .ADDRESS STS_VEC
RTS_REC: .LONG   0                              ;request to send
REC_NUM: .BLKB   6                              ;record-number
REC_DSC: .WORD   6
         .BYTE   DSC$K_DTYPE_T
         .BYTE   DSC$K_CLASS_S
         .ADDRESS REC_NUM
DATA_DSC:.WORD   0                              ;data-buffer-dsc
         .BYTE   DSC$K_DTYPE_T
         .BYTE   DSC$K_CLASS_D
         .ADDRESS 0
DATA_LEN:.LONG   0                              ;data-length
ND_NAME: .ASCID  /ALACK/                        ;node-name
AC_NAME: .ASCID  /CICSLU62/                     ;access-name
LUNAME : .ASCID  /TEST1/                        ;luname
TPN_NAM: .ASCID  /AIBR/                         ;transaction
                                                ;program name
WHAT_RC: .LONG   0                              ;message type
                                                ;received

EVENT_FL:.LONG   0                              ;event flag

         .PSECT  CODE,NOWRT,EXE,LONG
         .ENTRY  TEST_LU62, ^M<>                ;main program
                                                ;entry point

;
; Define LU name
;
         CLRL    R0
         PUSHAQ          AC_NAME                ;access name
         PUSHAQ  ND_NAME                        ;gateway node name
         PUSHL   #0                             ;security acceptance
         PUSHL   #0                             ;lu-lu password
         PUSHL   #0                             ;cnos support
         PUSHL   #0                             ;parallel session
                                                ;support
         PUSHL   #0                             ;initiate type
         PUSHL   #0                             ;uninterpreted luname
         PUSHAQ          LUNAME                 ;local luname
         PUSHAQ          LUNAME                 ;qualified luname
         PUSHAQ  STS_DSC                        ;status vector

         CALLS   #11,G^SNALU62$DEFINE_REMOTE    ;define a remote LU
         BLBS    R0, 10$
         BRW     EXITS                          ;exit if error
```

```
;
; Translate transaction program name to EBCDIC, allocate an event
; flag, and allocate a conversation
;
10$:     PUSHAQ  TPN_NAM                         ;transaction
         PUSHAQ  TPN_NAM                         ;program name

         CALLS   #2,G^LIB$TRA_ASC_EBC            ;translate data
                                                 ;to ebcdic

         PUSHAL  EVENT_FL                        ;efn

         CALLS   #1,G^LIB$GET_EF         ;allocate an event flag

         $CLREF_S -                              ;clear event flag
                 #5

         PUSHAL  EVENT_FL                        ;efn
         PUSHL   #0                              ;pip context
         PUSHL   #0                              ;profile
         PUSHL   #0                              ;password
         PUSHL   #0                              ;user id
         PUSHL   #0                              ;security
         PUSHAL  #SNALU62$K_SL_CONFIRM           ;sync-level
         PUSHAL  #SNALU62$K_WHEN_SESSION_ALLOC   ;return-control
         PUSHAL  #SNALU62$K_MAPPED_CONVERSATION ;conversation type
         PUSHAQ  TPN_NAM                         ;transaction
                                                 ;program name
         PUSHL   #0                              ;mode name
         PUSHAQ  LUNAME                          ;lu-name
         PUSHAL  #SNALU62$K_OTHER
         PUSHAQ  STS_DSC                         ;status-vector
         PUSHAL  RESC_ID                         ;resource id

         CALLS   #15,G^SNALU62$ALLOCATE          ;allocate conversation

         $WAITFR_S -                             ;wait for event flag
                 #5

         BLBS    R0, 20$
         BRW     EXITS                           ;exit if error

;
; Ask for record number, translate to EBCDIC and send data
;
20$:     PUSHAL  DATA_LEN                        ;data-length
         PUSHAQ  PROMPT                          ;prompt string
         PUSHAQ  REC_DSC                         ;record-number

         CALLS   #3,G^LIB$GET_INPUT              ;get record number

         PUSHAQ  REC_DSC                         ;translate
         PUSHAQ  REC_DSC                         ;record-number

         CALLS   #2,G^LIB$TRA_ASC_EBC            ; to ebcdic
```

```
         PUSHAL  RTS_REC                        ;request-to-send
                                                ;indicator
         PUSHAL  DATA_LEN                       ;data-length
         PUSHAQ  REC_DSC                        ;record-number
         PUSHAQ  STS_DSC                        ;status-vector
         PUSHAL  RESC_ID                        ;resource-id

         CALLS   #5,G^SNALU62$SEND_DATA         ;send data

         BLBS    R0, 30$
         BRW     EXITS                          ;exit if error

;
; Receive record from CICS/AIBR
;
30$:     MOVL    #^D<0>,DATA_LEN
         PUSHAL          WHAT_RC                ;message type
                                                ;received
         PUSHAQ  DATA_DSC                       ;data-buffer
         PUSHAL  RTS_REC                        ;request-to-send
                                                ;indicator
         PUSHAL  DATA_LEN                       ;data-length
         PUSHL   #0                             ;fill
         PUSHAQ  STS_DSC                        ;status-vector
         PUSHAL  RESC_ID                        ;resource-id

         CALLS   #7,G^SNALU62$RECEIVE_AND_WAIT  ;receive data

         BLBS    R0, 40$
         BRW     EXITS                          ;exit if error

;
; Convert record to ASCII and display on user terminal
;
40$:     PUSHAQ  DATA_DSC                       ;received
         PUSHAQ  DATA_DSC                       ;data-buffer

         CALLS   #2,G^LIB$TRA_EBC_ASC           ;translate data
                                                ;to ascii
         PUSHAQ  DATA_DSC
         CALLS   #1,G^LIB$PUT_OUTPUT            ;display data

;
; Receive DEALLOCATE
;
50$:     MOVL    #^D<0>,DATA_LEN
         PUSHAL          WHAT_RC                ;message type received
         PUSHAQ  DATA_DSC                       ;data-buffer
         PUSHAL  RTS_REC                        ;request-to-send
                                                ;indicator
         PUSHAL  DATA_LEN                       ;data-length
         PUSHL   #0                             ;fill
         PUSHAQ  STS_DSC                        ;status-vector
         PUSHAL  RESC_ID                        ;resource-id
```

```
        CALLS   #7,G^SNALU62$RECEIVE_AND_WAIT  ;receive deallocation
;
; Deallocate conversation
;
60$:    PUSHAL          #SNALU62$K_LOCAL       ;deallocation type
        PUSHAQ  STS_DSC                        ;status-vector
        PUSHAL  RESC_ID                        ;resource-id

        CALLS   #3,G^SNALU62$DEALLOCATE        ;deallocate

;
; Delete LU name
;
70$:    PUSHAQ          LUNAME                 ;qualified luname
        PUSHL   #0                             ;local luname
        PUSHAQ  STS_DSC                        ;status-vector

        CALLS   #3,G^SNALU62$DELETE            ;delete

EXITS:  $PUTMSG_S STS_VEC                      ;display
                                               ;status-vector
        $EXIT_S
        .END    TEST_LU62
```

COMMENTS

1. Assemble this MACRO program with a DCL command such as:

   ```
   $ MACRO/OBJECT=MYDIR:MYPROG SYS$LIBRARY:SNALU62DF+MYDIR:MYPROG
   ```

   where

   MYDIR and MYPROG are your directory and program.

2. You can use OpenVMS library routines to do parts of your application, such as translating ASCII to EBCDIC or vice versa.

3. An actual transaction program would check the *what_received* parameter. This example assumes that *what_received* equals SNALU62$K_DATA_ COMPLETE.

4. This example incorrectly handles the deallocate normally issued by the remote IBM host transaction program. The OpenVMS transaction program should check for the status code SNALU62$_DEALNOR and then deallocate the conversation.

5. Deactivate the session in a clean manner. Use the SNALU62$DELETE procedure to delete the remote LU defined with SNALU62$DEFINE_ REMOTE.

B.6 COBOL Programming Example

This COBOL program connects to AIBR, a sample transaction running under CICS. After establishing a connect with AIBR, the OpenVMS transaction program prompts you for the record number you want to send. This record number is converted to EBCDIC and sent to AIBR. The program receives data from the AIBR transaction, converts the data to ASCII format, and displays at your terminal.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TEST_LU62.
*****************************************************************
*                                                               *
* This program allocates a session with the AIBR transaction    *
* running under CICS 1.7.  It prompts the user for a record     *
* number, requests the record and displays it.  A deallocate    *
* message is sent and the session is terminated.  Proper operation *
* can be verified with SNATRACE.  This program is intended to   *
* represent the minimal framework for an SNA Gateway Application *
* Interface program.                                            *
*                                                               *
*****************************************************************

DATA DIVISION.

*
* Declaration
*
WORKING-STORAGE SECTION.

01  SS-STATUS               PIC S9(09) COMP.
01  I-STATUS                PIC S9(09) COMP.
01  RESOURCE-ID             PIC 9(08)  COMP.
01  STATUS-VEC              PIC X(64).
01  LUNAME                  PIC X(06)  VALUE "TEST1".
01  NODNAM                  PIC X(05)  VALUE "ALACK".
01  ACCNAM                  PIC X(08)  VALUE "CICSLU62".
01  ASCII-TPN-NAME          PIC X(04)  VALUE "AIBR".
01  TPN-NAME                PIC X(04).
01  EVENT_FLAG              PIC 9(08)  COMP.
01  WHAT-RECEIVED           PIC 9(08)  COMP.
01  RECORD-NUMBER           PIC X(06).
01  TEMP-DATA-BUFFER        PIC X(2000).
01  BUFFER-LENGTH           PIC 9(04)  VALUE 2000.
01  DATA-BUFFER             PIC X(2000).
01  DATA-LENGTH             PIC 9(04)  COMP.
01  TEMP-LENGTH             PIC 9(04).
01  RTS-REC                 PIC 9(04)  COMP.
01  OUTPUT-BUFFER.
        02 OUTPUT-BUF OCCURS 25 TIMES PIC X(80).
```

```
01  SUB                     PIC 9(03).
*
*
*
*****************************************************************
* SNA symbol definitions
*****************************************************************

01  SNALU62$_DEALNOR        PIC 9(08)  COMP VALUE 34832986.
01  SNALU62$K_OTHER         PIC 9(08)  COMP VALUE 2.
01  SNALU62$K_MAPPED_CONVERSATION
                            PIC 9(08)  COMP VALUE 4.
01  SNALU62$K_WHEN_SESSION_ALLOC
                            PIC 9(08)  COMP VALUE 5.
01  SNALU62$K_SL_CONFIRM    PIC 9(08)  COMP VALUE 9.
01  SNALU62$K_LOCAL         PIC 9(08)  COMP VALUE 32.

PROCEDURE DIVISION.
MAIN.
        PERFORM DEFINE-REMOTE.
        PERFORM ALLOCATE.
        PERFORM SEND-DATA.
        PERFORM RECEIVE-AND-WAIT.
        PERFORM TRANSLATE-EBC-ASC.
        PERFORM RECEIVE-AND-WAIT.
        PERFORM DEALLOCATE.
        PERFORM DELETE-PROC.
        PERFORM EXIT-PROGRAM.

*
* Define LU name
*
DEFINE-REMOTE.

        CALL "SNALU62$DEFINE_REMOTE" USING
                                BY DESCRIPTOR STATUS-VEC,
                                BY DESCRIPTOR LUNAME,
                                BY DESCRIPTOR LUNAME,
                                BY VALUE 0,0,0,0,0,0,
                                BY DESCRIPTOR NODNAM,
                                BY DESCRIPTOR ACCNAM,
                                        GIVING SS-STATUS.

        IF SS-STATUS IS FAILURE
          THEN
                PERFORM EXIT-PROGRAM.
```

```
*
* Translate transaction program name to EBCDIC and allocate
* conversation
*
ALLOCATE.
        CALL "LIB$TRA_ASC_EBC" USING BY DESCRIPTOR ASCII-TPN-NAME,
                                                    TPN-NAME,
                                           GIVING SS-STATUS.

        CALL "LIB$GET_EF" USING BY REFERENCE EVENT_FLAG GIVING I-STATUS.

        CALL "SYS$CLREF" USING BY VALUE EVENT_FLAG GIVING I-STATUS.
        CALL "SNALU62$ALLOCATE" USING
                        BY REFERENCE RESOURCE-ID,
                        BY DESCRIPTOR STATUS-VEC,
                        BY REFERENCE SNALU62$K_OTHER,
                        BY DESCRIPTOR LUNAME,
                        BY VALUE 0,
                        BY DESCRIPTOR TPN-NAME,
                        BY REFERENCE SNALU62$K_MAPPED_CONVERSATION,
                        BY REFERENCE SNALU62$K_WHEN_SESSION_ALLOC,
                        BY REFERENCE SNALU62$K_SL_CONFIRM,
                        BY VALUE 0,0,0,0,0,
                        BY REFERENCE EVENT_FLAG,
                                     GIVING SS-STATUS.
        CALL "SYS$WAITFR" USING BY VALUE EVENT_FLAG GIVING I-STATUS.

        IF SS-STATUS IS FAILURE
          THEN
                PERFORM EXIT-PROGRAM.

*
* Ask for record number, translate to EBCDIC and send data
*
SEND-DATA.

        DISPLAY "Enter record number: " WITH NO ADVANCING.
        ACCEPT RECORD-NUMBER.
        CALL "LIB$TRA_ASC_EBC" USING BY DESCRIPTOR RECORD-NUMBER,
                                           RECORD-NUMBER,
                                           GIVING SS-STATUS.
        MOVE 6 TO DATA-LENGTH.
        CALL "SNALU62$SEND_DATA" USING
                               BY REFERENCE RESOURCE-ID,
                               BY DESCRIPTOR STATUS-VEC,
                               BY DESCRIPTOR RECORD-NUMBER,
                               BY REFERENCE DATA-LENGTH,RTS-REC,
                                       GIVING SS-STATUS.
        IF SS-STATUS IS FAILURE
          THEN
                PERFORM EXIT-PROGRAM.
```

```
*
* Receive record from CICS/AIBR or DEALLOCATE
*
RECEIVE-AND-WAIT.
        MOVE BUFFER-LENGTH TO DATA-LENGTH.
        CALL "SNALU62$RECEIVE_AND_WAIT" USING
                                BY REFERENCE RESOURCE-ID,
                                BY DESCRIPTOR STATUS-VEC,
                                BY VALUE 0,
                                BY REFERENCE DATA-LENGTH,RTS-REC,
                                BY DESCRIPTOR DATA-BUFFER,
                                BY REFERENCE WHAT-RECEIVED,
                                        GIVING SS-STATUS.
        IF SS-STATUS IS FAILURE
          THEN
            IF NOT SS-STATUS = SNALU62$_DEALNOR
              THEN
                PERFORM EXIT-PROGRAM.
*
* Convert record to ASCII and display on user terminal
*
TRANSLATE-EBC-ASC.
        CALL "LIB$TRA_EBC_ASC" USING BY DESCRIPTOR DATA-BUFFER,
                                                   OUTPUT-BUFFER,
                                               GIVING SS-STATUS.
        DIVIDE 80 INTO DATA-LENGTH GIVING TEMP-LENGTH ROUNDED.
        ADD 1 TO TEMP-LENGTH.
        PERFORM DISPLAY-OUTPUT-BUFFER TEMP-LENGTH TIMES.
*
* Deallocate conversation
*
DEALLOCATE.
        CALL "SNALU62$DEALLOCATE" USING
                                BY REFERENCE RESOURCE-ID,
                                BY DESCRIPTOR STATUS-VEC,
                                BY REFERENCE SNALU62$K_LOCAL,
                                        GIVING SS-STATUS.
        IF SS-STATUS IS FAILURE
          THEN
                PERFORM EXIT-PROGRAM.
```

```
*
* Delete LU name
*
DELETE-PROC.
        CALL "SNALU62$DELETE" USING
                                  BY DESCRIPTOR STATUS-VEC,
                                  BY VALUE 0,
                                  BY DESCRIPTOR LUNAME,
                                          GIVING SS-STATUS.

EXIT-PROGRAM.
        CALL "SYS$PUTMSG" USING STATUS-VEC.
        STOP RUN.

DISPLAY-OUTPUT-BUFFER.
        IF SUB GREATER THAN 0
          THEN
            DISPLAY OUTPUT-BUF(SUB) WITH NO ADVANCING.
        ADD 1 TO SUB.
```

COMMENTS

1. You must define these symbols. Appendix F lists the symbols.
2. If you are going to send more data, the program should check the *rts_rec* parameter between calls.
3. An actual transaction program would check the *what_received* parameter. This example assumes that *what_received* equals SNALU62$K_DATA_ COMPLETE.
4. This example incorrectly handles the deallocate normal issued by the remote IBM host transaction program. The OpenVMS transaction program should check for the status code SNALU62$_DEALNOR and then deallocate the conversation.
5. You can use OpenVMS library routines to do parts of your application, such as translating ASCII to EBCDIC or vice versa.
6. Deactivate the session in a clean manner. Use the SNALU62$DELETE procedure to delete the remote LU defined with SNALU62$DEFINE_ REMOTE.

B.7 VAX PL/I Programming Example

This PL/I program connects to AIBR, a sample transaction running under CICS. After establishing a connect with AIBR, the OpenVMS transaction program prompts you for the record number you want to send. This record number is converted to EBCDIC and sent to AIBR. The program receives data from the AIBR transaction, converts the data to ASCII format, and displays at your terminal.

```
MAIN: PROCEDURE OPTIONS (MAIN) RETURNS (FIXED BINARY(31));
/*****************************************************************/
/*                                                               */
/* This program allocates a session with the AIBR transaction    */
/* running under CICS 1.7.  It prompts the user for a record     */
/* number, requests the record and displays it.  A deallocate    */
/* message is sent and the session is terminated.  Proper operation */
/* can be verified with SNATRACE.  This program is intended to   */
/* represent the minimal framework for an SNA Gateway Application */
/* Interface program.                                            */
/*                                                               */
/*****************************************************************/


/*****************************************************************/
/*             Declare External Routines first                   */
/*****************************************************************/

%INCLUDE $STSDEF;                       /* System status codes  */
%INCLUDE SYS$PUTMSG;                    /* System Services      */
%INCLUDE SYS$CLREF;
%INCLUDE SYS$WAITFR;
%INCLUDE 'SYS$LIBRARY:SNALU62DF.PLI';    /* SNALU62 symbols and */
                                         /* routine definitions */

DCL LIB$GET_INPUT EXTERNAL ENTRY (
            CHARACTER (*),                      /* Data          */
            CHARACTER (*),                      /* Prompt        */
            FIXED BIN (15))                     /* Size          */
    RETURNS (FIXED BIN(31));

DCL LIB$GET_EF EXTERNAL ENTRY (
            FIXED BIN (31))                      /* Event Flag    */
    RETURNS (FIXED BIN(31));

DCL LIB$PUT_OUTPUT EXTERNAL ENTRY (
            CHARACTER (*))                      /* Data          */
    RETURNS (FIXED BIN(31));
```

```
DCL LIB$TRA_ASC_EBC EXTERNAL ENTRY (
            CHARACTER (*),                      /* Input buffer  */
            CHARACTER (*))                      /* Output Buffer */
    RETURNS (FIXED BIN(31));

DCL LIB$TRA_EBC_ASC EXTERNAL ENTRY (
            CHARACTER (*),                      /* Input buffer  */
            CHARACTER (*))                      /* Output Buffer */
    RETURNS (FIXED BIN(31));

/*************************************************************/
/*              Declare variables and structures             */
/*************************************************************/

%REPLACE DATA_BUFFER_LENGTH BY 2000;

DECLARE NODE_NAME CHARACTER (5) INITIAL('ALACK'),

        ACCESS_NAME CHARACTER (8) INITIAL('CICSLU62'),

        RECORD_NUMBER CHARACTER (6),
        RECORD_NUMBER_SIZE FIXED BIN (15),
        RECORD_NUMBER_PROMPT CHARACTER (22)
                STATIC INITIAL('Enter record number: '),

        RESOURCE_ID FIXED BIN (31),
        DATA_SIZE FIXED BIN (15),
        STATUS_VECTOR CHARACTER (SNALU62$K_MIN_STATUS_VECTOR),

        LUNAME CHARACTER (6) INITIAL ('TEST1'),
        ASCII_TPN_NAME CHARACTER (4) INITIAL ('AIBR'),
        TPN_NAME CHARACTER (4),
        WHAT_RECEIVED FIXED BIN (31),
        SUB FIXED DECIMAL (3) INITIAL (1),
        INDEX FIXED DECIMAL (3) INITIAL (1),
        BUFFER_LENGTH FIXED BIN (15),
        DATA_BUFFER CHARACTER (DATA_BUFFER_LENGTH),
        RTS_REC FIXED BIN (31),
        EVENT_FLAG FIXED BIN (31) INITIAL(0);

/*****************************************************************/
/*      Define LU name                                           */
/*****************************************************************/

STS$VALUE = SNALU62$DEFINE_REMOTE(  STATUS_VECTOR,
                                    LUNAME,
                                    LUNAME,,,,,,,
                                    NODE_NAME,
                                    ACCESS_NAME);

IF ^STS$SUCCESS THEN GOTO ERROR_FROM_INTERFACE;

/*****************************************************************/
/*      Translate transaction program name to EBCDIC, allocate an   */
/* event flag, and allocate a conversation                       */
/*****************************************************************/
```

```
STS$VALUE = LIB$TRA_ASC_EBC (
                        ASCII_TPN_NAME,         /* transaction program name */
                        TPN_NAME
                        );
IF ^STS$SUCCESS THEN RETURN (STS$VALUE);

STS$VALUE = LIB$GET_EF( EVENT_FLAG );
IF ^STS$SUCCESS THEN RETURN (STS$VALUE);

STS$VALUE = SYS$CLREF( EVENT_FLAG );
STS$VALUE = SNALU62$ALLOCATE (
                                RESOURCE_ID,
                                STATUS_VECTOR,
                                SNALU62$K_OTHER,
                                LUNAME,
                                ,               /* name of remote LU */
                                TPN_NAME,
                                SNALU62$K_MAPPED_CONVERSATION,
                                SNALU62$K_WHEN_SESSION_ALLOC,
                                SNALU62$K_SL_CONFIRM,
                                ,,,,,   /* security,user_id,password,
                                           profile,pip_context*/
                                EVENT_FLAG);

IF ^STS$SUCCESS THEN GOTO ERROR_FROM_INTERFACE;
STS$VALUE = SYS$WAITFR( EVENT_FLAG );

/*****************************************************************/
/*    Ask for record number, translate to EBCDIC and send data      */
/*****************************************************************/

STS$VALUE = LIB$GET_INPUT (     RECORD_NUMBER,
                                RECORD_NUMBER_PROMPT,
                                RECORD_NUMBER_SIZE);
IF ^STS$SUCCESS THEN RETURN (STS$VALUE);

STS$VALUE = LIB$TRA_ASC_EBC (
                                RECORD_NUMBER,
                                RECORD_NUMBER
                                );
DATA_SIZE = 6;

STS$VALUE  =  SNALU62$SEND_DATA (
                                RESOURCE_ID,
                                STATUS_VECTOR,
                                RECORD_NUMBER,
                                DATA_SIZE,
                                RTS_REC,        /* request-to-send received */
                                ,               /* map name */
                                ,               /* fmh_data */
                                );

IF ^STS$SUCCESS THEN GOTO ERROR_FROM_INTERFACE;
```

```
/*****************************************************************/
/*      Receive record from CICS/AIBR                            */
/*****************************************************************/

DATA_SIZE = BUFFER_LENGTH;
STS$VALUE = SNALU62$RECEIVE_AND_WAIT (
                              RESOURCE_ID,
                              STATUS_VECTOR,
                              ,               /* fill */
                              DATA_SIZE,
                              RTS_REC,        /* request-to-send
                                                 received */
                              DATA_BUFFER,
                              WHAT_RECEIVED);

IF ^STS$SUCCESS THEN GOTO ERROR_FROM_INTERFACE;

/*****************************************************************/
/*      Convert record to ASCII and display on user terminal     */
/*****************************************************************/

STS$VALUE = LIB$TRA_EBC_ASC (
                        DATA_BUFFER,
                        DATA_BUFFER
                        );
IF ^STS$SUCCESS THEN RETURN (STS$VALUE);

STS$VALUE = LIB$PUT_OUTPUT (SUBSTR(DATA_BUFFER, 1, DATA_SIZE));
IF ^STS$SUCCESS THEN RETURN (STS$VALUE);

/*****************************************************************/
/*      Receive DEALLOCATE                                       */
/*****************************************************************/

STS$VALUE = SNALU62$RECEIVE_AND_WAIT (
                              RESOURCE_ID,
                              STATUS_VECTOR,
                              ,               /* fill */
                              DATA_SIZE,
                              RTS_REC,        /* request-to-send
                                                 received */
                              DATA_BUFFER,
                              WHAT_RECEIVED);

/*****************************************************************/
/*      Deallocate conversation                                  */
/*****************************************************************/

STS$VALUE = SNALU62$DEALLOCATE (
                              RESOURCE_ID,
                              STATUS_VECTOR,
                              SNALU62$K_LOCAL
                              );
```

```
/*****************************************************************/
/*      Delete LU name                                           */
/*****************************************************************/

STS$VALUE = SNALU62$DELETE (
                                STATUS_VECTOR,,          /* local LU name */
                                LUNAME
                                );

ERROR_FROM_INTERFACE:
STS$VALUE = SYS$PUTMSG (STATUS_VECTOR);
END;
```

COMMENTS

1. Include the LU6.2 library.
2. If you are going to send more data, the program should check the *rts_rec* parameter between calls.
3. An actual transaction program would check the *what_received* parameter. This example assumes that *what_received* equals SNALU62$K_DATA_ COMPLETE.
4. You can use OpenVMS library routines to do parts of your application, such as translating ASCII to EBCDIC or vice versa.
5. This example incorrectly handles the deallocate normal issued by the remote IBM host transaction program. The OpenVMS transaction program should check for the status code SNALU62$_DEALNOR and then deallocate the conversation.
6. Deactivate the session in a clean manner. Use the SNALU62$DELETE procedure to delete the remote LU defined with SNALU62$DEFINE_ REMOTE.

B.8 C Programming Examples

This C program connects to AIBR, a sample transaction running under CICS. After establishing a connect with AIBR, the OpenVMS transaction program prompts you for the record number you want to send. This record number is converted to EBCDIC and sent to AIBR. The program receives data from the AIBR transaction, converts the data to ASCII format, and displays at your terminal.

```
/*              LU62 Program                                            */

/*********************************************************************/
/*                                                                   */
/* This program allocates a session with the AIBR transaction        */
/* running under CICS 1.7.   It prompts the user for a record        */
/* number, requests the record and displays it.  A deallocate        */
/* message is sent and the session is terminated.  Proper operation */
/* can be verified with SNATRACE.  This program is intended to       */
/* represent the minimal framework for an SNA Gateway Application    */
/* Interface program.                                                */
/*                                                                   */
/*********************************************************************/

#include "sys$library:descrip.h"        /* Descriptor definitions   */
#include "sys$library:ssdef.h"          /* System services          */
#include "sys$library:stsdef.h"
#include "sys$library:snalu62df.h"              /* LU62 library     */

/*********************************************************************/
/* Declaration                                                       */
/*********************************************************************/

int
        status;

unsigned int
        status_vec[SNALU62$K_MIN_STATUS_VECTOR],
        remote_lu     = SNALU62$K_OTHER,
        convers_type  = SNALU62$K_MAPPED_CONVERSATION,
        return_cntrl  = SNALU62$K_WHEN_SESSION_ALLOC,
        sync_level    = SNALU62$K_SL_CONFIRM,
        dealloc_type  = SNALU62$K_LOCAL,
        event_flag    = 0,
        tpn_name[4],
        record_number[6],
        resource_id,
        what_received,
        rts_rec;

short unsigned int
        i,
        data_length,
        buffer_length;
```

```
struct  dsc$descriptor
        node_dsc,
        acc_name_dsc,
        luname_dsc,
        asc_tpn_dsc,
        prompt_dsc,
        status_vec_dsc,
        record_num_dsc,
        tpn_name_dsc,
        data_bufr_dsc,
        temp_bufr_dsc;

main( )
{

char *node_nm = "ALACK";
char *acc_nm  = "CICSLU62";
char *lu_nm   = "TEST1";
char *tpn_nm  = "AIBR";
char *prompt  = "Enter record number: ";
char *tpn_name = "    ";
char *space = " ";

/*****************************************************************/
/*      Initialize the descriptors                               */
/*****************************************************************/

status_vec_dsc.dsc$a_pointer = status_vec;
status_vec_dsc.dsc$w_length  = (SNALU62$K_MIN_STATUS_VECTOR);
status_vec_dsc.dsc$b_class   = 0;
status_vec_dsc.dsc$b_dtype   = 0;

record_num_dsc.dsc$a_pointer = record_number;
record_num_dsc.dsc$w_length  = (6);
record_num_dsc.dsc$b_class   = 0;
record_num_dsc.dsc$b_dtype   = 0;

tpn_name_dsc.dsc$a_pointer = tpn_name;
tpn_name_dsc.dsc$w_length  = (4);
tpn_name_dsc.dsc$b_class   = 0;
tpn_name_dsc.dsc$b_dtype   = 0;

data_bufr_dsc.dsc$a_pointer = (0);
data_bufr_dsc.dsc$w_length  = (0);
data_bufr_dsc.dsc$b_class   = DSC$K_CLASS_D;
data_bufr_dsc.dsc$b_dtype   = DSC$K_DTYPE_Z;

node_dsc.dsc$a_pointer = node_nm;
node_dsc.dsc$w_length  = (5);
node_dsc.dsc$b_class   = DSC$K_CLASS_S;
node_dsc.dsc$b_dtype   = DSC$K_DTYPE_Z;
```

```
acc_name_dsc.dsc$a_pointer = acc_nm;
acc_name_dsc.dsc$w_length  = (8);
acc_name_dsc.dsc$b_class   = DSC$K_CLASS_S;
acc_name_dsc.dsc$b_dtype   = DSC$K_DTYPE_Z;

luname_dsc.dsc$a_pointer = lu_nm;
luname_dsc.dsc$w_length  = (5);
luname_dsc.dsc$b_class   = DSC$K_CLASS_S;
luname_dsc.dsc$b_dtype   = DSC$K_DTYPE_Z;

asc_tpn_dsc.dsc$a_pointer = tpn_nm;
asc_tpn_dsc.dsc$w_length  = (4);
asc_tpn_dsc.dsc$b_class   = DSC$K_CLASS_S;
asc_tpn_dsc.dsc$b_dtype   = DSC$K_DTYPE_Z;

prompt_dsc.dsc$a_pointer = prompt;
prompt_dsc.dsc$w_length  = (22);
prompt_dsc.dsc$b_class   = DSC$K_CLASS_S;
prompt_dsc.dsc$b_dtype   = DSC$K_DTYPE_Z;

/*****************************************************************/
/* Define LU name                                                */
/*****************************************************************/

status = SNALU62$DEFINE_REMOTE(&status_vec_dsc,
                               &luname_dsc,
                               &luname_dsc,0,0,0,0,0,0,
                               &node_dsc,
                               &acc_name_dsc);

        if (!(status & STS$M_SUCCESS)) {
                        SYS$PUTMSG(&status_vec);
                        goto terminate;
          }

/*****************************************************************/
/* Translate transaction program name to EBCDIC, allocate an event  */
/* flag, and allocate the conversation                           */
/*****************************************************************/

status = LIB$TRA_ASC_EBC (&asc_tpn_dsc,
                          &tpn_name_dsc);

        if (!(status & STS$M_SUCCESS)) {
                        LIB$SIGNAL(status);
                        goto terminate;
          }

status = LIB$GET_EF (&event_flag);

        if (!(status & STS$M_SUCCESS)) {
                        SYS$PUTMSG(&status_vec);
                        goto terminate;
          }
```

```
status = SNALU62$ALLOCATE(&resource_id,
                          &status_vec_dsc,
                          &remote_lu,
                          &luname_dsc,
                          0,
                          &tpn_name_dsc,
                          &convers_type,
                          &return_cntrl,
                          &sync_level,
                          0,0,0,0,0,
                          &event_flag
                          );
        if (!(status & STS$M_SUCCESS)) {
                         SYS$PUTMSG(&status_vec);
                         goto terminate;
          }
status = SYS$WAITFR(event_flag);
        if (!(status & STS$M_SUCCESS)) {
                        LIB$SIGNAL(status);
          }
/*****************************************************************/
/* Ask for record number, translate to EBCDIC and send data      */
/*****************************************************************/
status = LIB$GET_INPUT (&record_num_dsc,
                         &prompt_dsc);
        if (!(status & STS$M_SUCCESS)) {
                         LIB$SIGNAL(status);
                         goto terminate;
          }
status = LIB$TRA_ASC_EBC (&record_num_dsc,
                          &record_num_dsc);
        if (!(status & STS$M_SUCCESS)) {
                         LIB$SIGNAL(status);
                         goto terminate;
          }
data_length = 6;
status = SNALU62$SEND_DATA (&resource_id,
                          &status_vec_dsc,
                          &record_num_dsc,
                          &data_length,
                          &rts_rec
                          );
        if (!(status & STS$M_SUCCESS)) {
                         SYS$PUTMSG(&status_vec);
                         goto terminate;
          }
```

```
/*****************************************************************/
/* Receive record from CICS/AIBR                                 */
/*****************************************************************/

data_length = buffer_length;

status = SNALU62$RECEIVE_AND_WAIT(&resource_id,
                                  &status_vec_dsc,
                                  0,
                                  &data_length,
                                  &rts_rec,
                                  &data_bufr_dsc,
                                  &what_received
                                  );

        if (!(status & STS$M_SUCCESS)) {
                        SYS$PUTMSG(&status_vec);
                        goto terminate;
          }
/*****************************************************************/
/* Convert record to ASCII and display on user terminal          */
/*****************************************************************/

status = LIB$TRA_EBC_ASC (&data_bufr_dsc,
                          &data_bufr_dsc);

        if (!(status & STS$M_SUCCESS)) {
                        LIB$SIGNAL(status);
                        goto terminate;
          }

status = LIB$PUT_OUTPUT (&data_bufr_dsc);

        if (!(status & STS$M_SUCCESS)) {
                        LIB$SIGNAL(status);
   goto terminate;
/*****************************************************************/
/* Receive DEALLOCATE                                            */
/*****************************************************************/

data_length = buffer_length;

status = SNALU62$RECEIVE_AND_WAIT(&resource_id,
                                  &status_vec_dsc,
                                  0,
                                  &data_length,
                                  &rts_rec,
                                  &data_bufr_dsc,
                                  &what_received
                                  );
```

```
        if (!(status & STS$M_SUCCESS)) {
                        if (status != SNALU62$_DEALNOR &&
                          LIB$SIGNAL(status);
                        else goto terminate;
          }
/*****************************************************************/
/* Deallocate conversation                                       */
/*****************************************************************/

terminate:
status = SNALU62$DEALLOCATE   ( &resource_id,
                                &status_vec_dsc,
                                &dealloc_type
                                );

        if (!(status & STS$M_SUCCESS)) {
                        SYS$PUTMSG(&status_vec);
          }

/*****************************************************************/
/* Delete LU name                                                */
/*****************************************************************/

status = SNALU62$DELETE( &status_vec_dsc,0,
                         &luname_dsc
                        );

exit(status);

}
```

COMMENTS

1. Include the LU6.2 library.
2. You can use OpenVMS library routines to do parts of your application, such as translating ASCII to EBCDIC or vice versa.
3. If you are going to send more data, the program should check the *rts_rec* parameter between calls.
4. An actual transaction program would check the *what_received* parameter. This example assumes that *what_received* equals SNALU62$K_DATA_ COMPLETE.
5. This example incorrectly handles the deallocate normal issued by the remote IBM host transaction program. The OpenVMS transaction program should check for the status code SNALU62$_DEALNOR and then deallocate the conversation.
6. Deactivate the session in a clean manner. Use the SNALU62$DELETE procedure to delete the remote LU defined with SNALU62$DEFINE_ REMOTE.

B.9 Second C Programming Example

This C program is functionally similar to the previous example. It establishes a conversation with AIBR and allows record numbers to be sent inbound. The specified records are received and displayed at your terminal. This program is a more realistic example of the state checking your program should do. It also demonstrates the use of notify routines to handle session failure and conversation deallocation events.

```
/*
**++
**  FACILITY:  LU62
**
**  MODULE DESCRIPTION:
**
**  Shows how to transmit a record key and get the record back from the
**  CICS AIBR transaction.  This program is similar to the previous 'C'
**  example but it provides some additional state checking.  It also
**  illustrates the use of the notify vectors on the ACTIVATE_SESSION and
**  ALLOCATE verbs.
**
**
**  CREATION DATE:
**
**      7/12/1990
**
**  DESIGN ISSUES:
**
**      This C program allocates a session with the AIBR transaction
**      running under CICS 1.7, It prompts the user for a 6 digit record
**      number, requests the record from AIBR and displays it.
**      This program illustrates some basic LU62 error handling.
**
**
**      The following outlines a typical data exchange flow with AIBR
**      ------------------------------------------------------------
**
**      local lu                                    remote lu  (CICS tran)
**      -------                                     ----------------------
**      define_remote
**      activate_session
**                              FMH5 attach to AIBR
**      allocate_conversation -------------------->CICS when receiving the FMH5
**                                                  will start the AIBR transaction
**                                                  AIBR will then wait for the
**                                                  record key to arrive, use this
**                                                  key to retrieve the record from
**                                                  the IBM data set and send this
**                                                  record back to Open VMS
**                                                  application.
```

```
**
**                                 record Key
**      send_data  --------------------------->  EXEC CICS RECEIVE INTO(buf)
**                                                              LENGTH(len)
**                                                    .
**                                                    .
**                                                <search for record>
**                                                    .
**      receive_and_wait <---------------------   EXEC CICS SEND FROM(record)
**      what_received= DATA                       LENGTH(length) WAIT LAST
**                                                EXEC CICS RETURN
**
**      Receive_and_wait <----------------------
**      status= DEALONOR
**
**      deallocate(local)
**         .
**         .
**    <now you can start another conversation, with another allocate as above>
**         .
**
**--
*/

#include <stdio.h>
#include <descrip.h>
#include <ssdef.h>
#include <stsdef.h>
#include <snalu62df.h>

/*
*  function prototypes
*/

unsigned int define_remote(char *,char *,char *,char *);
unsigned int activate_session(int *,char *,char *);
void         activate_session_notify_rtn(unsigned int *);
unsigned int allocate(int,int *,char *,char *,char *);
void         allocate_notify_rtn(unsigned int *);
unsigned int confirmed(int);
unsigned int send_data(int,char *);
unsigned int deallocate(int);
unsigned int receive_and_wait(int,char *,int *,int,int *);
unsigned int deallocate(int);
char         *get_record_key();
unsigned int process(unsigned int);
void         dump_cics_data(char *,unsigned short);
/*
* Type defintions
*/
#define cics_buffer 512
typedef char CICS_BUFFER[cics_buffer];
```

```
/*
*  Macro to build descriptor from a null terminated string
*/
#define DESCRIP(name,string) struct dsc$descriptor_s name= \
                             {strlen(string),DSC$K_DTYPE_T,DSC$K_CLASS_S,string}

/*
* Global variables
*/
static status_vec[SNALU62$K_MIN_STATUS_VECTOR];
static notify_vec[SNALU62$K_MIN_STATUS_VECTOR];
static struct dsc$descriptor status_vec_dsc;
static struct dsc$descriptor notify_vec_dsc;
static char *remote_lu_name = "remote";
static char *local_lu_name = "local";
static char *access_name = "CICS17B";
static char *remote_transaction = "AIBR";
static char *mode_name = "XL621024";
static char *gateway_name = "VINAL";


/*********************************************************************/
/*  main line                                                        */
/*********************************************************************/
main()
{
unsigned int session_id;
int status;

  /*
  Initialize the global status and notify vector descriptors
  */
  status_vec_dsc.dsc$a_pointer = &status_vec;
  status_vec_dsc.dsc$w_length = SNALU62$K_MIN_STATUS_VECTOR;
  status_vec_dsc.dsc$b_class = 0;
  status_vec_dsc.dsc$b_dtype = 0;

  notify_vec_dsc.dsc$a_pointer = &notify_vec;
  notify_vec_dsc.dsc$w_length = SNALU62$K_MIN_STATUS_VECTOR;
  notify_vec_dsc.dsc$b_class = 0;
  notify_vec_dsc.dsc$b_dtype = 0;

  /*
  * We now define a local alias name for the remote lu name that we can
  * use later when refereing to the remote lu
  */
  status= define_remote(remote_lu_name,local_lu_name,access_name,gateway_name);
  if(! (status&1))
     {
     LIB$SIGNAL(status);
     exit(1);
     }
```

```
  /*
  Activate the session
  */
  status = activate_session(&session_id,local_lu_name,mode_name);
  if(! (status&1))
     {
     LIB$SIGNAL(status);
     exit(1);
     }

  /*
  * Ask user for record numbers , and read the record from the CICS transaction
  * Interrupt when done
  */

  for(;;)
     if(! (1&(status = process(session_id))))
       {
        LIB$SIGNAL(status);
        exit(1);
       }

}
/*********************************************************************/
/*  process                                                          */
/*********************************************************************/
unsigned int
process(unsigned int session_id)
{
unsigned int conv_id;
char *key;
unsigned short len;
unsigned int what_received;
int status;
int read_again = TRUE;
enum STATE {START,SEND,RECEIVE,DEALLOCATE,CONFIRM,CONFIRM_SEND,
            CONFIRM_DEALLOCATE,ERROR,DONE};
enum STATE state;
CICS_BUFFER cics_data;

  state= START;

  while(state != DONE  && state != ERROR)
   switch(state)
     {
     case(START):
```

```
   status = allocate(session_id,
                  &conv_id,local_lu_name,mode_name,remote_transaction);
   if(! (status&1))
       state = ERROR;
    else
      {
      key = get_record_key();
      state = SEND;
      }

   break;

case(SEND):
   status = send_data(conv_id,key);
   if(! (1&status))
     switch(status)
       {
       case(SNALU62$_DEABPR):  /* the remote transaction Abended*/
       case(SNALU62$_DEABSVC):
       case(SNALU62$_DEABTIM):
       case(SNALU62$_RESFNO):
       case(SNALU62$_RESFRET):
          state = DEALLOCATE;
          break;
       case(SNALU62$_SVCERPU):
       case(SNALU62$_PRERPU):
          state = RECEIVE;
          break;
       default:
         state = ERROR;
         break;
      }
   else
     state = RECEIVE;

   break;
case(RECEIVE):
    status=receive_and_wait(conv_id
                           ,cics_data
                           ,&len
                           ,sizeof(cics_data)
                           ,&what_received);
```

```
 if(! (1&status))
   switch(status)
     {
     case(SNALU62$_PRERNTR):   /* program error no truncate */
     case(SNALU62$_SVCENTR):   /* service error no truncate */
     case(SNALU62$_PRERPU):    /* program error purging */
     case(SNALU62$_SVCERPU):   /* service error purging */
     case(SNALU62$_SVCERTR):   /* service error truncate */
     case(SNALU62$_PRERTR):    /* program error truncate */
        state = RECEIVE;        /* Stay in same state */
        break;

     case (SNALU62$_DEALNOR):   /* deallocate normal */
     case(SNALU62$_DEABPR):     /* deallocation abend program */
     case(SNALU62$_DEABSVC):    /* deallocation abend service */
     case(SNALU62$_DEABTIM):    /* deallocation abend timer */
     case(SNALU62$_ALLERR):     /* allocation error */
     case(SNALU62$_RESFNO):     /* resource failure no retry */
     case(SNALU62$_RESFRET):    /* resource failure retry */
        state = DEALLOCATE;
        break;

     default:
         state = ERROR;
         break;
     }
else
   switch(what_received)
     {
      case(SNALU62$K_DATA):
      case(SNALU62$K_DATA_COMPLETE):
      case(SNALU62$K_DATA_INCOMPLETE):
      case(SNALU62$K_LL_TRUNCATED):
        dump_cics_data(cics_data,len);
        state = RECEIVE;   /* keep same state */
        break;

     case(SNALU62$K_CONFIRM):
         state = CONFIRM;
         break;

     case(SNALU62$K_CONFIRM_SEND):
         state = CONFIRM_SEND;
         break;

     case(SNALU62$K_CONFIRM_DEALLOCATE):
         state = CONFIRM_DEALLOCATE;
         break;
```

```
            case(SNALU62$K_SEND):
                state = SEND;
                break;
            default:
                state = ERROR;  /*  this example only handles the above cases*/
                break;
            }
           break;

        case(CONFIRM_DEALLOCATE):
            status = confirmed(conv_id);
            if(! (1&status))
               state = ERROR;
            else
               state = DEALLOCATE;

            break;

        case(CONFIRM_SEND):
            status = confirmed(conv_id);
            if(! (1&status))
               state = ERROR;
            else
              state = SEND;

            break;

        case(CONFIRM):
            status = confirmed(conv_id);
            if(! (1&status))
               state = ERROR;
            else
              state = RECEIVE;

            break;

        case(DEALLOCATE):
           status = deallocate(conv_id);
           if(! (1&status))
              state = ERROR;
           else
              state = DONE;

           break;
       }
  return status;
}
```

```
/*********************************************************************/
/*  define_remote                                                    */
/*********************************************************************/
unsigned int
define_remote(char *remote_lu_name
             ,char *local_lu_name
             ,char *access_name
             ,char *gateway_name
             )
{
int status;
DESCRIP(remote_dsc,remote_lu_name);
DESCRIP(local_dsc,local_lu_name);
DESCRIP(access_dsc,access_name);
DESCRIP(gateway_dsc,gateway_name);
int session_number=53;
$DESCRIPTOR(circuit_dsc,"SNA-0");

    status= SNALU62$DEFINE_REMOTE(&status_vec_dsc
                                 ,&remote_dsc
                                 ,&local_dsc
                                 ,0
                                 ,0
                                 ,0
                                 ,0
                                 ,0
                                 ,0
                                 ,&gateway_dsc
                                 ,&access_dsc
                                 ,&circuit_dsc
                                 ,&session_number
                                 );

  if(! (status&1))
      SYS$PUTMSG(&status_vec);

  return status;
}

/*********************************************************************/
/*  activate_Session                                                 */
/*********************************************************************/
unsigned int
activate_session(int *session_id
                ,char *lu_name
                ,char *mode_name
                )
{
int status;
DESCRIP(lu_dsc,lu_name);
DESCRIP(mode_dsc,mode_name);
int polarity = SNALU62$K_BIDDER;
```

```
  status = SNALU62$ACTIVATE_SESSION(session_id
                                   ,&status_vec_dsc
                                   ,&lu_dsc
                                   ,&mode_dsc
                                   ,0
                                   ,0
                                   ,0
                                   ,activate_session_notify_rtn
                                   ,session_id
                                   ,&notify_vec_dsc
                                   ,&polarity
                                   );

  if(! (1&status))
    SYS$PUTMSG(&status_vec);

  return(status);

}

/*********************************************************************/
/*  allocate                                                         */
/*********************************************************************/
unsigned int
allocate(int  session_id
        ,int  *conv_id
        ,char *lu_name
        ,char *mode_name
        ,char *remote_transaction
        )
{
int status;
DESCRIP(lu_name_dsc,lu_name);
DESCRIP(mode_name_dsc,mode_name);
DESCRIP(remote_txn_dsc,remote_transaction);
int lu_type = SNALU62$K_OTHER;
int type =  SNALU62$K_MAPPED_CONVERSATION;
int ret_ctrl = SNALU62$K_WHEN_SESSION_ALLOC;
int sync_level = SNALU62$K_SL_NONE;
int security = SNALU62$K_NONE;
int polarity = SNALU62$K_BIDDER;
static short first_time = TRUE;

 if(first_time)  /* only  need to translate once the transaction name */
   {
   status= LIB$TRA_ASC_EBC(&remote_txn_dsc,&remote_txn_dsc);
   if(! (status&1))
     return status;

   first_time = FALSE;
   }
```

```
 status= SNALU62$ALLOCATE(conv_id
                         ,&status_vec_dsc
                         ,&lu_type
                         ,&lu_name_dsc
                         ,&mode_name_dsc
                         ,&remote_txn_dsc
                         ,&type
                         ,&ret_ctrl
                         ,&sync_level
                         ,&security
                         ,0,0,0,0,0,0,0
                         ,allocate_notify_rtn
                         ,&session_id
                         ,&polarity);

  if(! (1&status))
    SYS$PUTMSG(&status_vec);

  return(status);

}
/*********************************************************************/
/*  confirmed                                                        */
/*********************************************************************/
unsigned int
confirmed(int conv_id)
{
int status;

    status = SNALU62$CONFIRMED(&conv_id
                              ,&status_vec_dsc);

   if(! (1&status))
     SYS$PUTMSG(&status_vec);

   return(status);

}
/*********************************************************************/
/*  send_data                                                        */
/*********************************************************************/
unsigned int
send_data(int conv_id
         ,char *key
         )
{
int status;
int fmh_data= FALSE;    /* no Function managment included in this RU */
DESCRIP(data,key);

    status= LIB$TRA_ASC_EBC(&data,&data);
    if(! (status&1))
       return status;
```

```
    status= SNALU62$SEND_DATA(&conv_id
                              ,&status_vec_dsc
                              ,&data
                              ,0
                              ,0
                              ,0
                              ,&fmh_data
                              ,0,0,0);

  if(! (1&status))
    SYS$PUTMSG(&status_vec);

  return status;

}

/*********************************************************************/
/*  deallocate                                                       */
/*********************************************************************/
unsigned int
deallocate(int conv_id)
{
int status;
int type= SNALU62$K_LOCAL;

  status= SNALU62$DEALLOCATE(&conv_id
                             ,&status_vec_dsc
                             ,&type
                             );

  if(! (1&status))
    SYS$PUTMSG(&status_vec);

  return status;

}

/*********************************************************************/
/*  receive_and_wait                                                 */
/*********************************************************************/
unsigned int
receive_and_wait(int conv_id
                ,char *into
                ,int  *actual_length
                ,int  max_avaliable
                ,int  *what_rec
                )
{
int status;
int fill = SNALU62$K_LL;
int len;
int rts_rec;
$DESCRIPTOR(data,"");
```

```
    data.dsc$w_length = max_avaliable;
    data.dsc$a_pointer = into;

    status= SNALU62$RECEIVE_AND_WAIT(&conv_id
                                    ,&status_vec_dsc
                                    ,&fill
                                    ,actual_length
                                    ,&rts_rec
                                    ,&data
                                    ,what_rec
                                    );

  if(! (1&status))
    SYS$PUTMSG(&status_vec);

   return status;
}

/*********************************************************************/
/*  activate_session_notify_rtn                                      */
/*********************************************************************/
void activate_session_notify_rtn(unsigned int *notify_parameter)
/*
AST triggered when no verb is outstanding and the session fails
*/
{

 printf("Session %d has failed\n"
        ,*notify_parameter);

 /*
 Display the reason for the failure
 */
 SYS$PUTMSG(&notify_vec);

 /* Here also you may want to set some global flag to notify the
    non-ast executing thread of the fact that the session has been terminated.
 */
}

/*********************************************************************/
/*  allocate_notify_rtn                                              */
/*********************************************************************/
void allocate_notify_rtn(unsigned int *notify_parameter)
/*
AST triggered when no verb is outstanding and deallocate is received
*/
{

 printf("Current conversation received a deallocate message on session %8x\n"
        ,*notify_parameter);
```

```
 /* Here also you may want to set some global flag to notify the
    non-ast executing thread of the fact that the conversation has
    received a deallocate request.  A conversation verb should then be
    issued.  The conversation is not in DEALLOCATE state until the
    what_received parameter so indicates.
 */

}

/*****************************************************************/
/*  dump_cics_data                                               */
/*****************************************************************/
void
dump_cics_data(char *data,unsigned short len)
{
int status;
$DESCRIPTOR(data_dsc,"");

  data_dsc.dsc$w_length = len;
  data_dsc.dsc$a_pointer = data;

  status= LIB$TRA_EBC_ASC(&data_dsc,&data_dsc);
  status= LIB$PUT_OUTPUT(&data_dsc);

}
/*****************************************************************/
/*  get_record_key                                               */
/*****************************************************************/
char *
get_record_key()
{
static char buf[BUFSIZ];
int i;

  /*
  * Our sample CICS transaction AIBR requires 6 digit key number
  */
  printf("Enter record number to read from CICS (6 number key required) >");
  i= scanf("%s",buf);
  while(i != 1)
   {
   fflush(stdin);
   printf("retry again, Enter record number >");
   i = scanf("%s",buf);
   }

return buf;

}
```

B.10 Third C Programming Example

This C program demonstrates the use of contention loser polarity. A single session is established by both inbound and outbound conversations. If there is contention for the session, the OpenVMS transaction will lose and the allocate request will fail.

```
/*
**++
**  FACILITY:  LU62
**
**  MODULE DESCRIPTION:
**
**   Shows how to handle contention resolution.
**
**  CREATION DATE:
**
**      7/12/1990
**
**  DESIGN ISSUES:
**
**      We specify that our lu be the contention loser on a conversation
**      allocation conflict. If we attempt to allocate a conversation
**      at the same time as the remote lu, using the same session,
**      our allocate request will fail.
**
**
**
**  ABSTRACT:
**
** 1.  Defines the remote LU IMSBRIDGE
** 2.  Defines a TPN IMSASYNC
**      3.  Attempts to allocate an inbound  conversation
**     with IMS /TEST.  We will be the contention loser so
**     if an outbound conversation for IMSASYNC is initiated,
**     our BID will be rejected and the ALLOCATE request will fail.
** 4.  Receives data from the host for either inbound and outbound
**     conversations.
** 5.  For inbound conversations, data can be sent to the host and
**     echoed back.
** 6.  The conversation is deallocated locally and the LU is deleted.
**
**--
*/

#include <stdio.h>
#include <descrip.h>
#include <ssdef.h>
#include <stsdef.h>
#include <snalu62df.h>
```

```
/*
*  function prototypes
*/

unsigned int define_remote(char *,char *,char *,char *,char *);
unsigned int allocate(int *,char *,char *,char *);
unsigned int confirmed(int);
unsigned int send_data(int,char *);
unsigned int deallocate(int);
unsigned int receive_and_wait(int,char *,unsigned short *,int,int *);
unsigned int prepare_to_receive(int);
unsigned int delete_lu(char *,char *,char *);
char         *get_user_input();
unsigned int process();
void         dump_IMS_data(char *,unsigned short);
void attach_routine();                    /* ast for FMH attach */
unsigned int define_tp(char *);
unsigned int is_it_bid_rejection();

/*
* Type definitions
*/

#define ims_buffer 512
typedef char IMS_BUFFER[ims_buffer];

/*
* Macro to build descriptor from a null terminated string
*/

#define DESCRIP(name,string) struct dsc$descriptor_s name = \
                             {strlen(string),DSC$K_DTYPE_T,DSC$K_CLASS_S,string}

/*
* Global variables
*/

static status_vec[SNALU62$K_MIN_STATUS_VECTOR];
static struct dsc$descriptor status_vec_dsc;
static char *remote_lu_name = "remote";
static char *local_lu_name = "local";

static char *tpn_name ="IMSASYNC";
static char *plu_name ="SE40BRDG";
static char *remote_transaction = "/TEST";
static char *mode_name = "XL621024";
static char *gateway_name = "VINAL";
static int  ibm_to_vax_session_id;
static int  IMS_event_flag;
```

```
/*************************************************************/
/* main                                                      */
/*************************************************************/
main()
{
unsigned int session_id;
int status;

  status_vec_dsc.dsc$a_pointer = &status_vec;
  status_vec_dsc.dsc$w_length =  SNALU62$K_MIN_STATUS_VECTOR;
  status_vec_dsc.dsc$b_class =0;
  status_vec_dsc.dsc$b_dtype =0;

  /*
  * Define the LU name
  */

  status = define_remote(remote_lu_name
                      ,local_lu_name
                      ,gateway_name,plu_name,mode_name);
  if(! (status&1))
     {
     LIB$SIGNAL(status);
     exit(1);
     }

  /*
  * Define the TP name IMSASYNC
  */

  status = define_tp(tpn_name);
  if(! (status&1))
     {
     LIB$SIGNAL(status);
     exit(1);
     }

  /*
  * Get an event flag to synchronize with ATTACH FMH arrival
  */

  status = LIB$GET_EF(&IMS_event_flag);
  if(! (status&1))
     {
     LIB$SIGNAL(status);
     exit(1);
     }
```

```
  /*
  * Process the conversation
  */
  if(! (1&(status = process())))
     LIB$SIGNAL(status);
  else
     {
     /*
     * Delete the LU name resources
     */

     status = delete_lu(local_lu_name,remote_lu_name,tpn_name);

     if(! (status&1))
         LIB$SIGNAL(status);
     }

  exit(1);

}

/************************************************************/
/*  process                                                 */
/************************************************************/
unsigned int
process()
{
unsigned int conv_id;
char *key;
unsigned short len;
unsigned int what_received;
int status;
int read_again = TRUE;
enum STATE {START,SEND,RECEIVE,DEALLOCATE,CONFIRM,CONFIRM_SEND,
            CONFIRM_DEALLOCATE,PREPARE_TO_RECEIVE,ERROR,DONE};
enum STATE state;
IMS_BUFFER ims_data;

  state = START;

  while(state != DONE  && state != ERROR)
   switch(state)
     {
     case(START):

 /*
 *  Attempt to allocate a conversation
 */

        status = allocate(&conv_id,local_lu_name,mode_name,remote_transaction);

        if(! (status&1))
```

```
/*
*  If the allocation has failed, check for a BIDREJ
*  error.  If this is the case, deallocate the
*  conversation locally and wait for the pending
*  outbound conversation.
*/

      if(is_it_bid_rejection()) /* check if it is a contention loss */
         {
         status = deallocate(conv_id);

         if(! (status&1))
            state = ERROR;
         else
            {
            printf("\n IMS wants to transmit first, waiting.. \n");

            status = SYS$WAITFR(IMS_event_flag);
            if(! (1&status))
              state = ERROR;
            else
               {
               conv_id = ibm_to_vax_session_id;  /* use the current conv */
               state = RECEIVE;
               }
            }
         }
      else
        state = ERROR;
    else
       state = RECEIVE;  /* now receive the DFS0581 TEST COMMAND COMPLETE*/
                        /* Response from IMS */

   break;
```

```
case(SEND):
   key = get_user_input();
   status = send_data(conv_id,key);
   if(! (1&status))
     switch(status)
       {
       case(SNALU62$_DEABPR):  /* the remote transaction Abended*/
       case(SNALU62$_DEABSVC):
       case(SNALU62$_DEABTIM):
       case(SNALU62$_RESFNO):
       case(SNALU62$_RESFRET):
          state = DEALLOCATE;
          break;
       case(SNALU62$_SVCERPU):
       case(SNALU62$_PRERPU):
          state = PREPARE_TO_RECEIVE;
          break;
       default:
         state = ERROR;
         break;
      }
   else
     state = RECEIVE;

   break;

case(PREPARE_TO_RECEIVE):
    status = prepare_to_receive(conv_id);
    if(! (status&1))
      state = ERROR;
    else
       state = RECEIVE;
    break;

case(RECEIVE):
    status = receive_and_wait(conv_id
                             ,ims_data
                             ,&len
                             ,sizeof(ims_data)
                             ,&what_received);
```

```
 if( !(1&status) )
   switch(status)
     {
      case(SNALU62$_PRERNTR):   /* program error no truncate */
      case(SNALU62$_SVCENTR):   /* service error no truncate */
      case(SNALU62$_PRERPU):    /* program error purging */
      case(SNALU62$_SVCERPU):   /* service error purging */
      case(SNALU62$_SVCERTR):   /* service error truncate */
      case(SNALU62$_PRERTR):    /* program error truncate */
         state = RECEIVE;        /* Stay in same state */
         break;
     case (SNALU62$_DEALNOR):   /* deallocate normal */
     case(SNALU62$_DEABPR):     /* deallocation abend program */
     case(SNALU62$_DEABSVC):    /* deallocation abend service */
     case(SNALU62$_DEABTIM):    /* deallocation abend timer */
     case(SNALU62$_ALLERR):     /* allocation error */
     case(SNALU62$_RESFNO):     /* resource failure no retry */
     case(SNALU62$_RESFRET):    /* resource failure retry */
        state = DEALLOCATE;
        break;
     default:
         state = ERROR;
         break;
     }
else
   switch(what_received)
     {
      case(SNALU62$K_DATA):
      case(SNALU62$K_DATA_COMPLETE):
      case(SNALU62$K_DATA_INCOMPLETE):
      case(SNALU62$K_LL_TRUNCATED):
        dump_IMS_data(ims_data,len);
        state = RECEIVE;   /* keep same state */

        break;

     case(SNALU62$K_CONFIRM):
         state = CONFIRM;
         break;

     case(SNALU62$K_CONFIRM_SEND):
         state = CONFIRM_SEND;
         break;

     case(SNALU62$K_CONFIRM_DEALLOCATE):
         state = CONFIRM_DEALLOCATE;
         break;
```

```
            case(SNALU62$K_SEND):
                state = SEND;
                break;
            default:
                state = ERROR;  /*  this example only handles the above cases*/
                break;
            }
           break;

        case(CONFIRM_DEALLOCATE):
            status = confirmed(conv_id);
            if(! (1&status))
               state = ERROR;
            else
              state = DEALLOCATE;

            break;

        case(CONFIRM_SEND):
            status = confirmed(conv_id);
            if(! (1&status))
               state = ERROR;
            else
              state = SEND;

            break;

        case(CONFIRM):
            status = confirmed(conv_id);
            if(! (1&status))
               state = ERROR;
            else
              state = RECEIVE;

            break;

        case(DEALLOCATE):
           status = deallocate(conv_id);
           if(! (1&status))
              state = ERROR;
           else
              state = DONE;

           break;
       }

  return status;
}
```

```
/*************************************************************/
/*  Define_remote                                            */
/*************************************************************/
unsigned int
define_remote(char *remote_lu_name
              ,char *local_lu_name
              ,char *gateway_name
              ,char *plu_name
              ,char *logon_name
              )
{
int status;
DESCRIP(remote_dsc,remote_lu_name);
DESCRIP(local_dsc,local_lu_name);
DESCRIP(gateway_dsc,gateway_name);
DESCRIP(plu_dsc,plu_name);
DESCRIP(logon_dsc,logon_name);
DESCRIP(circuit_dsc,"SNA-0");
int init_only = SNALU62$K_INITIATE_ONLY;
int session_number = 53;

    status= SNALU62$DEFINE_REMOTE(&status_vec_dsc
                                 ,&remote_dsc
                                 ,&local_dsc
                                 ,0
                                 ,&init_only
                                 ,0
                                 ,0
                                 ,0
                                 ,0
                                 ,&gateway_dsc
                                 ,0
                                 ,&circuit_dsc
                                 ,&session_number
                                 ,&plu_dsc
                                 ,&logon_dsc
                                 );

  if(! (status&1)) SYS$PUTMSG(&status_vec);

  return status;
}
```

```
/************************************************************/
/*  Delete_lu                                               */
/************************************************************/
unsigned int
delete_lu(char *local_lu
         ,char *remote_lu
         ,char *tpn_name)
{
DESCRIP(local_lu_dsc,local_lu);
DESCRIP(remote_lu_dsc,remote_lu);
DESCRIP(tpn_name_dsc,tpn_name);
int status;

     status= SNALU62$DELETE   (&status_vec_dsc
                              ,&local_lu_dsc
                              ,&remote_lu_dsc
                              ,0
                              ,&tpn_name_dsc
                              );

  if(! (1&status))
    SYS$PUTMSG(&status_vec);

  return(status);
}

/************************************************************/
/*  Allocate                                                */
/************************************************************/
unsigned int
allocate(int  *conv_id
        ,char *lu_name
        ,char *mode_name
        ,char *remote_transaction
        )
{
int status;
DESCRIP(lu_name_dsc,lu_name);
DESCRIP(mode_name_dsc,mode_name);
DESCRIP(remote_txn_dsc,remote_transaction);
int lu_type = SNALU62$K_OTHER;
int type =  SNALU62$K_MAPPED_CONVERSATION;
int ret_ctrl = SNALU62$K_WHEN_SESSION_ALLOC;
int sync_level = SNALU62$K_SL_CONFIRM;
int security = SNALU62$K_NONE;
int polarity = SNALU62$K_BIDDER;
static short first_time = TRUE;
```

```
 if(first_time)  /* only  need to translate once the transaction name */
   {
   status = LIB$TRA_ASC_EBC(&remote_txn_dsc,&remote_txn_dsc);
   if(! (status&1))
     return status;
   first_time = FALSE;
   }

 /*
 * We use polarity as SNALU62$K_BIDDER meaning that if IMS wishes to
 * allocate a conversation also , we lose the contention for the
 * session
 */

 status = SNALU62$ALLOCATE(conv_id
                          ,&status_vec_dsc
                          ,&lu_type
                          ,&lu_name_dsc
                          ,0
                          ,&remote_txn_dsc
                          ,&type
                          ,&ret_ctrl
                          ,&sync_level
                          ,0
                          ,0,0,0,0,0,0,0,0,0
                          ,&polarity);

  if(! (1&status))
    SYS$PUTMSG(&status_vec);

  return(status);
}


/*************************************************************/
/*  Respond with Confirmed                                   */
/*************************************************************/
unsigned int
confirmed(int conv_id)
{
int status;

    status = SNALU62$CONFIRMED(&conv_id
                              ,&status_vec_dsc);

   if(! (1&status))
     SYS$PUTMSG(&status_vec);

   return(status);

}
```

```
/************************************************************/
/*  Send Data                                               */
/************************************************************/
unsigned int
send_data(int conv_id
         ,char *key
         )
{
int status;
int fmh_data = FALSE;    /* no Function managment included in this RU */
DESCRIP(data,key);

    status= LIB$TRA_ASC_EBC(&data,&data);
    if(! (status&1))
       return status;

    status = SNALU62$SEND_DATA(&conv_id
                              ,&status_vec_dsc
                              ,&data
                              ,&data.dsc$w_length
                              ,0
                              ,0
                              ,&fmh_data
                              ,0,0,0);

  if(! (1&status))
    SYS$PUTMSG(&status_vec);

  return status;

}
/************************************************************/
/*  Deallocate the local LU resources                       */
/************************************************************/

unsigned int
deallocate(int conv_id)
{
int status;
int type= SNALU62$K_LOCAL;

  status = SNALU62$DEALLOCATE(&conv_id
                             ,&status_vec_dsc
                             ,&type
                             );

  if( !(1&status) )
    SYS$PUTMSG(&status_vec);

  return status;

}
```

```
/************************************************************/
/*  Receive data from the host                              */
/************************************************************/
unsigned int
receive_and_wait(int conv_id
                ,char *into
                ,unsigned short *actual_length
                ,int  max_avaliable
                ,int  *what_rec
                )
{
int status;
int fill = SNALU62$K_LL;
int len;
int rts_rec;
$DESCRIPTOR(data,"");

    data.dsc$w_length = max_avaliable;
    data.dsc$a_pointer = into;

    status = SNALU62$RECEIVE_AND_WAIT(&conv_id
                                     ,&status_vec_dsc
                                     ,&fill
                                     ,actual_length
                                     ,&rts_rec
                                     ,&data
                                     ,what_rec
                                     );

  if( !(1&status) )
    SYS$PUTMSG(&status_vec);

   return status;
}

/************************************************************/
/*  Define the TP name INSASYNC                             */
/************************************************************/

unsigned int
define_tp(char *tp)
{
int status;
DESCRIP(tp_dsc,tp);

   status = LIB$TRA_ASC_EBC(&tp_dsc,&tp_dsc);
   if(! (status&1))
     return status;
```

```
   status = SNALU62$DEFINE_TP(&status_vec_dsc
                             ,&tp_dsc
                             ,0,0,0,0,0,0,0,0,0,0
                             ,attach_routine
                             );

  if( !(1&status) )
    SYS$PUTMSG(&status_vec);

   return status;
}


/*************************************************************/
/*  Prepare to receive                                       */
/*************************************************************/

unsigned int
prepare_to_receive(int conv)
{
int status;

    status = SNALU62$PREPARE_TO_RECEIVE(&conv
                                       ,&status_vec_dsc
                                       );


  if( !(1&status) )
    SYS$PUTMSG(&status_vec);

   return status;
}

/*************************************************************/
/*  Attach routine for outbound conversations                */
/*************************************************************/

void
attach_routine(int *parm, int *id)   /* AST level */
{

 printf("\n IMSASYNC started by bridge \n");
 ibm_to_vax_session_id = *id;

 SYS$SETEF(IMS_event_flag);

}

/*************************************************************/
/*  Display any data received                                */
/*************************************************************/
```

```
void
dump_IMS_data(char *data,unsigned short len)
{
int status;
$DESCRIPTOR(data_dsc,"");

  data_dsc.dsc$w_length = len;
  data_dsc.dsc$a_pointer = data;

  status= LIB$TRA_EBC_ASC(&data_dsc,&data_dsc);

  status= LIB$PUT_OUTPUT(&data_dsc);
  printf("\n");

}

/************************************************************/
/*  Get data from the terminal                              */
/************************************************************/

char *
get_user_input()
{
static char buf[BUFSIZ];
int i;

  printf("Enter Data to send to IMS >");
  i = scanf("%s",buf);
  while(i != 1)
   {
   fflush(stdin);
   printf("retry again, Enter record number >");
   i = scanf("%s",buf);
   }

return buf;

}

/************************************************************/
/*  See if this is a SNALU62$_BIDREJ error.  It will be     */
/*  in the third longword of the status vector.  It is      */
/*  preceded by SNALU62$_RESFRET.        */
/************************************************************/
unsigned int
is_it_bid_rejection()
{

 /*
 * Check the status vector for the error  SNALU62$_BIDREJ
 */
    if(status_vec[3] == SNALU62$_BIDREJ) return TRUE;

  return FALSE;
}
```

C

Return Codes and State Changes

C.1 Return Codes and State Changes for Conversations

Following is a list of top-level status codes returned by a conversation verb, and the state change, if any, that a conversation has undergone.

Return Code	State Change
SNALU62$_ALLERR	The conversation is in deallocate state.
SNALU62$_DEABPR	The conversation is in deallocate state.
SNALU62$_DEABSVC	The conversation is in deallocate state.
SNALU62$_DEABTIM	The conversation is in deallocate state.
SNALU62$_DEALNOR	The conversation is in deallocate state.
SNALU62$_FMHNOT	The conversation is in send state.
SNALU62$_OK	The state of the conversation is as defined for the procedure.
SNALU62$_PARERR	The state of the conversation remains unchanged.
SNALU62$_PRERNTR	The conversation is in receive state.
SNALU62$_PRERPU	The conversation is in receive state.
SNALU62$_PRERTR	The conversation is in receive state.
SNALU62$_RESFNO	The conversation is in deallocate state.
SNALU62$_RESFRET	The conversation is in deallocate state.
SNALU62$_SVCENTR	The conversation is in receive state.
SNALU62$_SVCERPU	The conversation is in receive state.
SNALU62$_SVCERTR	The conversation remains in receive state.
SNALU62$_STAERR	The conversation state remains unchanged.
SNALU62$_UNSUC	The conversation state remains unchanged.

C.2 Return Codes for Control Operator Verbs

Following is a table of top-level status codes that can be returned by a control operator verb.

Return Code
SNALU62$_ASNEG
SNALU62$_ASSPEC
SNALU62$_OK
SNALU62$_PARERR
SNALU62$_RESFNO
SNALU62$_RESFRET
SNALU62$_STAERR
SNALU62$_UNSUC

D

Conversation State Transitions

This appendix shows the conversation state transitions that can occur when a program issues a conversation verb. A conversation enters a particular state when either a transaction program issues a verb that causes a state transition, or a transaction program receives a return code that indicates state transition.

Specific state transitions are defined in individual verb descriptions under the heading "State Transition," and in return code descriptions under the heading "Status Codes."

Figure D–1 correlates each conversation verb to the conversation state allowing its issuance. The columns of the table show the individual conversation state. The rows show the individual verbs. A verb is shown more than once when a parameter of the verb or return code determines the state transitions that can occur.

The conversation states are:

Reset	Transaction program can allocate the conversation.
Send	Transaction program can send data.
Receive	Transaction program can receive information from the remote program.
Confirm	Transaction program can respond to a confirmation request.

Confirm_Deallocate

Confirm_Send — Confirm_Deallocate and Confirm_Send are pseudo-states. They are used in this table to distinguish events that take you into the confirm state.

Deallocate — Transaction program can deallocate the conversation locally.

Note

Version 2.4 of the APPC/LU6.2 Programming Interface does not support sync level syncpt.

D.1 Using the Conversion State Transitions Table

Refer to row 1 of Figure D–1 for this example.

Assume no conversation exists, and the state is SNALU62$K_STATE_RESET (column 1). Issuing the SNALU62$ALLOCATE verb and receiving a return code of SNALU62$K_OK will cause the conversation to transition to SNALU62$K_STATE_SEND (column 2).

Figure D–1 Conversion State Transitions

Verb \ State	1 SNALU62$K_STATE_RESET	2 SNALU62$K_STATE_SEND	3 SNALU62$K_STATE_DEFER_RECEIVE	4 SNALU62$K_STATE_DEFER_DEALLOC	5 SNALU62$K_STATE_RECEIVE	6 SNALU62$K_STATE_CONFIRM	7 SNALU62$K_STATE_CONFIRM_SEND	8 SNALU62$K_STATE_CONFIRM_DEALLOC	9 SNALU62$K_STATE_SYNCPT	10 SNALU62$K_STATE_SYNCPT_SEND	11 SNALU62$K_STATE_SYNCPT_DEALLOC	12 SNALU62$K_STATE_DEALLOC
Initiating Conversation	5	/	/	/	/	/	/	/	/	/	/	/
SNALU62$ALLOCATE[ok]	2	/	/	/	/	/	/	/	/	/	/	/
SNALU62$ALLOCATE[ae]	12	/	/	/	/	/	/	/	/	/	/	/
SNALU62$ALLOCATE[pe]	–	/	/	/	/	/	/	/	/	/	/	/
SNALU62$ALLOCATE[un]	–	/	/	/	/	/	/	/	/	/	/	/
SNALU62$CONFIRM[ok]	/	–	5	1	/	/	/	/	/	/	/	/
SNALU62$CONFIRM[ae]	/	12	12	12	/	/	/	/	/	/	/	/
SNALU62$CONFIRM[bo]	/	1	1	1	/	/	/	/	/	/	/	/
SNALU62$CONFIRM[dn]	/	12	12	12	/	/	/	/	/	/	/	/
SNALU62$CONFIRM[da]	/	5	5	5	/	/	/	/	/	/	/	/
SNALU62$CONFIRM[rl]	/	12	12	12	/	/	/	/	/	/	/	/
SNALU62$CONFIRMED	/	/	/	/	/	5	2	12	/	/	/	/
SNALU62$DEALLOCATE(F)[ok]	/	1	/	/	/	/	/	/	/	/	/	/
SNALU62$DEALLOCATE(C)[ok]	/	1	/	/	/	/	/	/	/	/	/	/
SNALU62$DEALLOCATE(A)[ok]	/	1	1	1	1	1	1	1	1	1	1	/
SNALU62$DEALLOCATE(L)[ok]	/	/	/	/	/	/	/	/	/	/	/	1
SNALU62$DEALLOCATE(C)[ae]	/	12	/	/	/	/	/	/	/	/	/	/
SNALU62$DEALLOCATE(C)[da]	/	12	/	/	/	/	/	/	/	/	/	/
SNALU62$DEALLOCATE(C)[ep]	/	5	/	/	/	/	/	/	/	/	/	/
SNALU62$DEALLOCATE(C)[rf]	/	12	/	/	/	/	/	/	/	/	/	/
SNALU62$FLUSH	/	–	5	1	/	/	/	/	/	/	/	/
SNALU62$GET_ATTRIBUTES	/	–	–	–	–	–	–	–	–	–	–	–
SNALU62$GET_TYPE	/	–	–	–	–	–	–	–	–	–	–	–
SNALU62$POST_ON_RECIEPT	/	/	/	/	–	/	/	/	/	/	/	/
SNALU62$PREPARE_TO_RECEIVE(F)[ok]	/	5	/	/	/	/	/	/	/	/	/	/
SNALU62$PREPARE_TO_RECEIVE(C)[ok]	/	5	/	/	/	/	/	/	/	/	/	/
SNALU62$PREPARE_TO_RECEIVE(C)[ae]	/	12	/	/	/	/	/	/	/	/	/	/
SNALU62$PREPARE_TO_RECEIVE(C)[da]	/	12	/	/	/	/	/	/	/	/	/	/
SNALU62$PREPARE_TO_RECEIVE(C)[ep]	/	5	/	/	/	/	/	/	/	/	/	/
SNALU62$PREPARE_TO_RECEIVE(C)[rf]	/	12	/	/	/	/	/	/	/	/	/	/

(continued on next page)

Figure D–1 (Cont.) Conversion State Transitions

Verb \ State	1 SNALU62$K_STATE_RESET	2 SNALU62$K_STATE_SEND	3 SNALU62$K_STATE_DEFER_RECEIVE	4 SNALU62$K_STATE_DEFER_DEALLOC	5 SNALU62$K_STATE_RECEIVE	6 SNALU62$K_STATE_CONFIRM	7 SNALU62$K_STATE_CONFIRM_SEND	8 SNALU62$K_STATE_CONFIRM_DEALLOC	9 SNALU62$K_STATE_SYNCPT	10 SNALU62$K_STATE_SYNCPT_SEND	11 SNALU62$K_STATE_SYNCPT_DEALLOC	12 SNALU62$K_STATE_DEALLOC
SNALU62$RECEIVE_AND_WAIT[ok]{dt}	/	5	/	/	–	/	/	/	/	/	/	/
SNALU62$RECEIVE_AND_WAIT[ok]{se}	/	–	/	/	2	/	/	/	/	/	/	/
SNALU62$RECEIVE_AND_WAIT[ok]{co}	/	6	/	/	6	/	/	/	/	/	/	/
SNALU62$RECEIVE_AND_WAIT[ok]{cs}	/	7	/	/	7	/	/	/	/	/	/	/
SNALU62$RECEIVE_AND_WAIT[ok]{cd}	/	8	/	/	8	/	/	/	/	/	/	/
SNALU62$RECEIVE_AND_WAIT[ae]	/	12	/	/	12	/	/	/	/	/	/	/
SNALU62$RECEIVE_AND_WAIT[bo]	/	1	/	/	1	/	/	/	/	/	/	/
SNALU62$RECEIVE_AND_WAIT[da]	/	12	/	/	12	/	/	/	/	/	/	/
SNALU62$RECEIVE_AND_WAIT[dn]	/	12	/	/	12	/	/	/	/	/	/	/
SNALU62$RECEIVE_AND_WAIT[en]	/	5	/	/	–	/	/	/	/	/	/	/
SNALU62$RECEIVE_AND_WAIT[ep]	/	5	/	/	–	/	/	/	/	/	/	/
SNALU62$RECEIVE_AND_WAIT[et]	/	/	/	/	–	/	/	/	/	/	/	/
SNALU62$RECEIVE_AND_WAIT[rl]	/	12	/	/	12	/	/	/	/	/	/	/
SNALU62$RECEIVE_IMMEDIATE[ok]{da}	/	/	/	/	–	/	/	/	/	/	/	/
SNALU62$RECEIVE_IMMEDIATE[ok]{se}	/	/	/	/	2	/	/	/	/	/	/	/
SNALU62$RECEIVE_IMMEDIATE[ok]{co}	/	/	/	/	6	/	/	/	/	/	/	/
SNALU62$RECEIVE_IMMEDIATE[ok]{cs}	/	/	/	/	7	/	/	/	/	/	/	/
SNALU62$RECEIVE_IMMEDIATE[ok]{cd}	/	/	/	/	8	/	/	/	/	/	/	/
SNALU62$RECEIVE_IMMEDIATE[ae]	/	/	/	/	12	/	/	/	/	/	/	/
SNALU62$RECEIVE_IMMEDIATE[bo]	/	/	/	/	–	/	/	/	/	/	/	/
SNALU62$RECEIVE_IMMEDIATE[da]	/	/	/	/	12	/	/	/	/	/	/	/
SNALU62$RECEIVE_IMMEDIATE[dn]	/	/	/	/	12	/	/	/	/	/	/	/
SNALU62$RECEIVE_IMMEDIATE[en]	/	/	/	/	–	/	/	/	/	/	/	/
SNALU62$RECEIVE_IMMEDIATE[ep]	/	/	/	/	–	/	/	/	/	/	/	/
SNALU62$RECEIVE_IMMEDIATE[et]	/	/	/	/	–	/	/	/	/	/	/	/
SNALU62$RECEIVE_IMMEDIATE[rf]	/	/	/	/	12	/	/	/	/	/	/	/
SNALU62$RECEIVE_IMMEDIATE[un]	/	/	/	/	–	/	/	/	/	/	/	/
SNALU62$REQUEST_TO_SEND	/	/	/	/	–	/	–	/	–	/	/	/
SNALU62$SEND_DATA[ok]	/	–	/	/	/	/	/	/	/	/	/	/

(continued on next page)

Figure D–1 (Cont.) Conversion State Transitions

Verb	1 SNALU62$K_STATE_RESET	2 SNALU62$K_STATE_SEND	3 SNALU62$K_STATE_DEFER_RECEIVE	4 SNALU62$K_STATE_DEFER_DEALLOC	5 SNALU62$K_STATE_RECEIVE	6 SNALU62$K_STATE_CONFIRM	7 SNALU62$K_STATE_CONFIRM_SEND	8 SNALU62$K_STATE_CONFIRM_DEALLOC	9 SNALU62$K_STATE_SYNCPT	10 SNALU62$K_STATE_SYNCPT_SEND	11 SNALU62$K_STATE_SYNCPT_DEALLOC	12 SNALU62$K_STATE_DEALLOC
SNALU62$SEND_DATA[ae]	/	12	/	/	/	/	/	/	/	/	/	/
SNALU62$SEND_DATA[bo]	/		/	/	/	/	/	/	/	/	/	/
SNALU62$SEND_DATA[da]	/	12	/	/	/	/	/	/	/	/	/	/
SNALU62$SEND_DATA[ep]	/	5	/	/	/	/	/	/	/	/	/	/
SNALU62$SEND_DATA[rf]	/	12	/	/	/	/	/	/	/	/	/	/
SNALU62$SEND_ERROR[ok]	/	–	/	/	2	2	2	2	2	2	2	/
SNALU62$SEND_ERROR[ae]	/	12	/	/	/	/	/	/	/	/	/	/
SNALU62$SEND_ERROR[bo]	/		/	/	/	/	/	/	/	/	/	/
SNALU62$SEND_ERROR[da]	/	12	/	/	/	/	/	/	/	/	/	/
SNALU62$SEND_ERROR[dn]	/	/	/	/	12	/	/	/	/	/	/	/
SNALU62$SEND_ERROR[ep]	/	5	/	/	/	/	/	/	/	/	/	/
SNALU62$WAIT[ok]	/	/	/	/	–	/	/	/	/	/	/	/
SNALU62$WAIT[ae]	/	/	/	/	12	/	/	/	/	/	/	/
SNALU62$WAIT[bo]	/	/	/	/	1	/	/	/	/	/	/	/
SNALU62$WAIT[da]	/	/	/	/	12	/	/	/	/	/	/	/
SNALU62$WAIT[ok]	/	/	/	/	12	/	/	/	/	/	/	/
SNALU62$WAIT[en]	/	/	/	/	–	/	/	/	/	/	/	/
SNALU62$WAIT[ep]	/	/	/	/	–	/	/	/	/	/	/	/
SNALU62$WAIT[et]	/	/	/	/	–	/	/	/	/	/	/	/
SNALU62$WAIT[pn]	/	/	/	/	–	/	/	/	/	/	/	/
SNALU62$WAIT[rf]	/	/	/	/	12	/	/	/	/	/	/	/

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)

What-Received Abbreviations {...}:

co CONFIRM
cd CONFIRM_DEALLOCATE
cs CONFIRM_SEND
dt DATA, DATA_COMPLETE, DATA_INCOMPLETE, or LL_TRUNCATED
se SEND

Matrix Symbols:

/ Verb cannot be issued in this state.
– Remain in current state.
number Number of next state.

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
ok OK
pe PARAMETER_ERROR
pn POSTING_NOT_ACTIVE
rf RESOURCE_FAILURE_NO_RETRY or RESOURCE_FAILURE_RETRY
un UNSUCCESSFUL

E

APPC/LU6.2 Interface Procedures and Status Messages

Figure E–1 illustrates the correlation between the APPC/LU6.2 Programming Interface procedures and the status messages they can return.

Figure E–1 Status Messages

Status Message	ACTIVATE_SESSION	ALLOCATE	CONFIRM	CONFIRMED	DEACTIVATE_SESSION	DEALLOCATE	DEFINE_REMOTE	DEFINE_TP	DELETE	FLUSH	GET_ATTRIBUTES	GET_PIP	GET_TYPE	POST_ON_RECEIPT	PREPARE_TO_RECEIVE	RECEIVE_AND_WAIT	RECEIVE_IMMEDIATE	REQUEST_TO_SEND	SEND_DATA	SEND_ERROR	SUPPLY_PIP	WAIT
SNALU62$_OK		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
SNALU62$_ALLERR		X	X											X	X			X	X			
SNALU62$_ASNEG	X																					
SNALU62$_ASSPEC	X																					
SNALU62$_DEABPR			X											X	X	X		X	X			
SNALU62$_DEABSVC			X											X	X	X		X	X			
SNALU62$_DEABTIM			X											X	X	X		X	X			
SNALU62$_DEALNOR															X	X			X			
SNALU62$_FMHNOT			X											X	X	X		X	X			
SNALU62$_MAPEFAI			X											X	X	X		X	X			
SNALU62$_MAPNFOU			X											X	X	X		X	X			

(continued on next page)

Figure E–1 (Cont.) Status Messages

Status Message	ACTIVATE_SESSION	ALLOCATE	CONFIRM	CONFIRMED	DEACTIVATE_SESSION	DEALLOCATE	DEFINE_REMOTE	DEFINE_TP	DELETE	FLUSH	GET_ATTRIBUTES	GET_PIP	GET_TYPE	POST_ON_RECEIPT	PREPARE_TO_RECEIVE	RECEIVE_AND_WAIT	RECEIVE_IMMEDIATE	REQUEST_TO_SEND	SEND_DATA	SEND_ERROR	SUPPLY_PIP	WAIT
SNALU62$_PARERR	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
SNALU62$_PRERNTR															X	X						
SNALU62$_PRERPU			X											X	X	X		X	X			
SNALU62$_PRERTR														X	X	X						
SNALU62$_RESFNO	X		X												X	X		X	X			
SNALU62$_RESFRET	X	X	X											X	X	X		X	X			
SNALU62$_STAERR		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
SNALU62$_SVCENTR															X	X						
SNALU62$_SVCERPU			X											X	X	X		X	X			
SNALU62$_SVCERTR															X	X						
SNALU62$_UNSUC	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

F

Definitions for the APPC/LU6.2 Interface

The following table presents symbols, values, and meanings to use when you write your OpenVMS transaction program. DIGITAL recommends that you use the definition files that accompany the APPC/LU6.2 Interface. This will insulate you from changes made in future releases of the product. Definition files, however, are not provided for every language. If the language you plan to use does not have a definition file, use the information in the following table to write your application.

Symbol	Value	Meaning
SNALU62$K_ABEND_PROG	29	Flush and deallocate abnormally when in send or defer state
SNALU62$K_ABEND_SVC	30	Flush and deallocate abnormally when in send or defer state
SNALU62$K_BASIC_CONVERSATION	3	Allocate a basic conversation
SNALU62$K_BUFFER	35	Receive unformatted data
SNALU62$K_CONFIRM	44	Remote program issued confirm
SNALU62$K_CONFIRM_DEALLOCATE	46	Remote program issued deallocate confirm
SNALU62$K_CONFIRM_SEND	45	Remote program issued prepare to receive, then confirm
SNALU62$K_DATA	37	Unformatted data transferred
SNALU62$K_DATA_COMPLETE	38	Complete logical record transferred
SNALU62$K_DATA_INCOMPLETE	39	Incomplete logical record transferred
SNALU62$K_DEACT_CLEANUP	73	Deactivate cleanup
SNALU62$K_DEACT_NORMAL	72	Deactivate normal

Symbol	Value	Meaning
SNALU62$K_FLUSH	28	Flush and deallocate conversation
SNALU62$K_FMH_DATA_COMPLETE	40	Complete logical record with FMH
SNALU62$K_FMH_DATA_INCOMPLETE	41	Incomplete logical record with FMH
SNALU62$K_INITIATE_ONLY	57	Request session activation
SNALU62$K_INITIATE_OR_QUEUE	58	Wait for session activation
SNALU62$K_LL	36	Receive one logical record
SNALU62$K_LL_TRUNCATED	42	LL count truncated after first byte
SNALU62$K_LOCAL	32	Deallocate conversation locally
SNALU62$K_LONG	34	Return when information received
SNALU62$K_MAPPED_CONVERSATION	4	Allocate a mapped conversation
SNALU62$K_MIN_NOTIFY_VECTOR	64	16 longwords = 64 bytes
SNALU62$K_MIN_STATUS_VECTOR	64	16 longwords = 64 bytes
SNALU62$K_NONE	54	Omit access security information on allocation request
ET91FCH4.SDML		
SNALU62$K_OTHER	2	Remote program is located at another LU
SNALU62$K_OWN	1	Remote program exists at local LU
SNALU62$K_PGM	56	Use access security information supplied on allocation request
SNALU62$K_PROG	51	Application program error being reported
SNALU62$K_SAME	55	Use access security information from previous allocation request
SNALU62$K_SEND	43	Remote program entered receive state
SNALU62$K_SHORT	33	Return when affirmative reply received
SNALU62$K_SL_CONFIRM	9	Perform confirmation processing

Symbol	Value	Meaning
SNALU62$K_SL_NONE	8	No confirmation or sync-point processing
SNALU62$K_STATE_CONFIRM	6	Conversation is in confirm state
SNALU62$K_STATE_CONFIRM_ DEALLOC	8	Conversation is in confirm deallocate state
SNALU62$K_STATE_CONFIRM_SEND	7	Conversation is in confirm send state
SNALU62$K_STATE_DEALLOCATE	12	Conversation is in deallocate state
SNALU62$K_STATE_RECEIVE	5	Conversation is in receive state
SNALU62$K_STATE_RESET	1	Conversation is in reset state
SNALU62$K_STATE_SEND	2	Conversation is in send state
SNALU62$K_SVC	52	LU services error being reported
SNALU62$K_SYNC_LEVEL	27	Deallocate based on conversation sync level
SNALU62$K_WHEN_SESSION_ALLOC	5	Return control when session is allocated

G

APPC/LU6.2 Programming Interface Compatibility Features

The features described in this appendix, available with Versions 1.0 and 1.1 of the DIGITAL SNA APPC/LU6.2 Programming Interface for OpenVMS, are supported by Version 2.4 only for compatibility with those earlier versions. New programs should not be written with these procedures.

The calling format for the APPC/LU6.2 procedures referenced in this appendix, is in Chapter 4 of this manual.

G.1 The SNALU62$DEFINE Procedure

The SNALU62$DEFINE procedure defines various entities that are associated with a logical unit.

The parameter list consists of a *function* parameter to specify the entity to be defined, followed by one or more parameter pairs with the format

par id, *par*

where *par id* is a symbolic name identifying the parameter and *par* is the actual parameter.

To undefine an LU name, call the SNALU62$DEFINE procedure and specify the value SNALU62$K_LUNAME and the name of the LU name to be undefined. Do not specify any other parameters. Any active sessions that currently have no conversations on them and were started with this LU name are immediately deactivated.

If a conversation is currently on a particular session referenced in the SNALU62$DEFINE procedure call, the session will be deactivated when the conversation is deallocated.

Note

SNALU62$DEFINE has been superseded by the following procedures:

- SNALU62$DEFINE_REMOTE
- SNALU62$DELETE

Note parameter changes on each verb.

Format:

status.wlc.v=SNALU62$DEFINE	(*status blk.wx.dx*), *function* *[par1 id.rlu.r,par1 rx.dx*], [*par2 id.rlu.r,par2 rx.dx*], . . . [*parN id.rlu.r, parN rx.dx*]

Arguments:

status	When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits, together, represent the severity of the error. Returned by function value.
status_blk	A longword vector allocated by the OpenVMS transaction program and filled in by the APPC/LU6.2 Programming Interface to provide the user with complete status information. Passed by descriptor.
function	The following symbolic name: SNALU62$K_ DEFINE_LUNAME causes the APPC/LU6.2 Programming Interface to define a local name for a remote LU. Passed by reference.
par1_id,par1	The identifier SNALU62$K_LUNAME followed by a local name for the remote LU. This name is an ASCII string. The number of characters in this string is not limited. The identifier is passed by reference. The local name is passed by descriptor.
par2 id, par2 . . 3 *parN_id,parN*	An identifier followed by an attribute of the LU specified in *par1*. The identifier/attribute pairs are listed in Table G-1. The identifier is passed by reference. The attribute is passed by descriptor.

Table G–1 LU Attribute Identifiers and Attributes

Identifier	Attribute
SNALU62$K_ GWYNODE	The name of the SNA gateway OpenVMS SNA node. This parameter is an ASCII string. For OpenVMS SNA (OpenVMS VAX Version 6.1 and Version 6.2 only), set this parameter equal to an ASCII 0. If the gateway node name is not supplied, the APPC/LU6.2 Programming Interface assumes you are requesting a connection by means of OpenVMS SNA.
SNALU62$K_ ACCNAME	The gateway access name used in setting up the session. This parameter is an ASCII string of from 1 to 8 characters. If this parameter is omitted or if the access name definition does not include all the required information, access information must be provided by means of the remaining parameter pairs described in this table. For complete information on access names, see the *DIGITAL SNA Peer Server Management* and *DIGITAL SNA Domain Gateway Management*.
SNALU62$K_CIRCUIT	The gateway circuit name. This parameter is an ASCII string of the form *dev-n*; for example, SDP-1 for the DX24 Gateway, LC-0 for the DECSA Gateway, or SNA-0 for OpenVMS/SNA.
SNALU62$K_SESADDR	The gateway secondary logical unit (SLU) address. This parameter is an integer in the range of 1 through 255. Although this attribute is an integer value, the APPC/LU6.2 Programming Interface requires that the parameter be passed by descriptor. Most languages will default to passing the parameter by reference, so you must override the default mechanism.
SNALU62$K_PLU	The VTAM name for the primary logical unit (PLU). This parameter is an ASCII string of from 1 to 8 characters.
SNALU62$K_LOGON	A logon mode table entry name defined at the remote IBM host. This parameter is an ASCII string of from 1 to 8 characters.
SNALU62$K_USER_ DATA	A string up to 128 characters long that may be needed to complete the logon to the PLU. This character string is not translated to EBCDIC.

G.1.1 Status Codes

The SNALU62$DEFINE procedure can return the following status codes:

- SNALU62$_OK
- SNALU62$_PARERR

- SNALU62$_UNSUC

G.1.2 State Transition

None.

G.2 Status Codes

The following list contains obsolete error messages that a procedure returns in the status vector.

UDEFPAR, unrecognized DEFINE parameter (parameter *parameter number*, value *parameter id*)

Explanation: The indicated parameter has the invalid value shown by the message.

User Action: Correct the call to SNALU62$DEFINE.

UFUNCOD, unrecognized DEFINE function code (value *number*)

Meaning: The function code supplied on the SNALU62$DEFINE procedure is invalid.

User Action: Supply a valid function code to SNALU62$DEFINE.

G.3 Obsolete Definitions for the APPC/LU6.2 Programming Interface

The following table presents symbols, values and meanings you may have used with early versions of the APPC/LU6.2 Programming Interface.

Symbol	Value	Meaning
SNALU62$K_ACCNAME	17	Gateway access name
SNALU62$K_CIRCUIT	18	Gateway circuit name
SNALU62$K_DEFINE_ LUNAME	13	Define an LUNAME
SNALU62$K_GWYNODE	16	SNA gateway nodename
SNALU62$K_LU_NAME	15	Local LUNAME being defined
SNALU62$K_PLU	20	The VTAM name for the primary logical unit
SNALU62$K_SESADDR	19	Gateway secondary logical unit address

Symbol	Value	Meaning
SNALU62$K_USER_DATA	22	A string of up to 128 EBCDIC characters

H

Status Codes

Status messages that are returned by the DIGITAL SNA APPC/LU6.2 Programming Interface for OpenVMS, as well as the lower layers of SNA software, are displayed at your terminal in the following format:

```
%facility-l-ident, text
```

where

facility is the SNALU62$ or the SNA$ component name. A percent sign (%) prefixes the first message displayed on your screen, and a hyphen (-) prefixes each subsequent message.

l is the severity level indicator. It has one of the following values:

S indicates success. The system performed your request; your procedure completed without failure.

I indicates information. The system performed your request; your command completed without failure. Information about the circumstances under which the operation completed is included.

W indicates warning. Warning messages indicate that the command may have performed part, but not all, of your request. You need to verify the command or program output.

E indicates error. Error messages are top-level errors that a procedure returns to the status vector. They indicate conditions that prevent a procedure from completing successfully. Additional suberror messages may be displayed explaining why the operation failed.

F indicates fatal. Fatal messages indicate that the system cannot continue execution of the request. You cannot recover from a fatal error. You must try to correct the condition that is causing the error.

ident is an abbreviation of the message text.

text is the explanation of each status code.

Both the facility name and severity level indicator have been removed from the messages listed in this appendix. Messages are listed alphabetically by *ident*. A status message is displayed on your screen as follows:

```
%SNALU62$-E-ALLBLK, failed to allocate internal data structure
```

The following status messages are returned by the DIGITAL SNA APPC/LU6.2 Programming Interface:

ABNSESTER, session terminated abnormally

Facility: SNA$

Explanation: Either the link between the gateway and IBM was lost or IBM deactivated the physical unit (PU) or the line leading to the gateway.

User Action: Determine why the link was lost. Retry when the connection to IBM returns.

ACCINTERR, Gateway detected an error in the gateway access routines

Facility: SNA$

Explanation: A fatal error has occurred.

User Action: Copy the error messages that appear on your screen at this time and report the problem to your system manager.

ALLBLK, failed to allocate internal data structure

Facility: SNALU62$

Explanation: The APPC/LU6.2 Programming Interface failed to allocate memory for an internal data structure. The most likely reason is that no free virtual memory was available.

User Action: If you are using class D descriptors, make sure you return used buffers to free memory using LIB$SFREE1_DD or STR$FREE1_DX. If your process uses a lot of virtual memory, you may need to increase the PGFLQUOTA parameter for that user by means of the AUTHORIZE utility. See the appropriate OpenVMS System Management Guide for your system for details on how to use this utility.

ALLERR, conversation could not be allocated

Facility: SNALU62$

Explanation: The remote LU rejected the allocation request. Subsequent error messages identify specific errors. The error may be returned on procedure calls other than SNALU62$ALLOCATE because of the way the APPC/LU6.2 Programming Interface uses internal buffers (For more

information about this, see the *IBM Transaction Programmer's Reference Manual for LU Type 6.2*, Order No. GC30-3084).

User Action: Examine the subsequent error messages, check and adjust OpenVMS transaction program logic, then rerun the program.

APPNOTSPE, IBM application name was not specified

Facility: SNA$

Explanation: In the connect request, you did not specify the IBM application name, and the access name that you used did not supply one either.

User Action: The IBM application must be either explicitly supplied in the parameter list or implicitly supplied through the access name.

ASIINV, access security information is invalid

Facility: SNALU62$

Explanation: The user is not allowed to access the transaction program.

User Action: Have your IBM system manager add a user ID, password, and profile to the list of allowable users. Check the call to SNALU62$ALLOCATE to see if it has specified the correct conversation level security, user ID, password, and profile.

ASNEG, bind image was negotiated

Facility: SNALU62$

Explanation: The bind image was negotiated; the session was activated as specified.

User Action: No response required. Successful completion code.

ASSPEC, bind image accepted as specified

Facility: SNALU62$

Explanation: The bind image was accepted; the session was activated after negotiation.

User Action: No response required. Successful completion code.

BACKOUT, backout

Facility: SNALU62$

Explanation: The transaction program has backed out of the conversation.

User Action: Examine the subsequent error messages, check and adjust OpenVMS transaction program logic, then rerun the program.

BIDREJ, Bid request was rejected

Facility: SNALU62$

Explanation: The IBM host has rejected the bid request of the OpenVMS transaction.

User Action: Retry the SNALU62$ALLOCATE request. This error can only occur if the OpenVMS transaction is the contention loser. Refer to the *polarity* parameter for SNALU62$ALLOCATE.

BINSPEUNA, the BIND image specified unacceptable values

Facility: SNA$

Explanation: The gateway rejected the BIND request image.

User Action: Run a trace to find out why the gateway rejected the BIND request. The IBM application could be specifying too large an outbound RU or an illegal function management (FM) or transmission services (TS) profile, or it could have sent a pacing value that was out of bounds (see the appropriate installation guide for your transport).

BUFSHO, data length specified exceeds size of supplied buffer

Facility: SNALU62$

Explanation: The *length* parameter on the SNALU62$SEND_DATA procedure is greater than the size of the buffer specified by the *data* parameter.

User Action: Correct the call to the SNALU62$SEND_DATA procedure.

BUGCHK, internal consistency failure

Facility: SNALU62$

Explanation: An internal software error has been detected.

User Action: The appearance of this error message implies that the APPC/LU6.2 Programming Interface is operating abnormally. To take corrective action, copy all the messages associated with the fatal error code and take them to your DIGITAL system manager, who will decide what action to take. If you still cannot solve your problem, submit a Software Performance Report (SPR), or call the DIGITAL support hotline.

CONREQREJ, connect request rejected by IBM host sense code *IBM sense code*

Facility: SNA$

Explanation: The IBM host rejected the connect request for the reason given in the sense code.

User Action: Determine the meaning of the sense code from the IBM documentation and take the appropriate action.

CONTMIS, conversation type mismatch

Facility: SNALU62$

Explanation: The remote LU rejected the conversation allocation request because the remote IBM host transaction program does not support the requested conversation type. Conversation types are mapped and basic.

User Action: Specify a conversation type on the SNALU62$ALLOCATE procedure that the remote IBM host transaction program does support and reissue the procedure.

DEABPR, deallocate abend program

Facility: SNALU62$

Explanation: The remote IBM host transaction program has aborted, thereby ending the conversation. Subsequent messages identify specific errors. The conversation is in deallocate state.

User Action: Examine subsequent error messages, then see your IBM systems programmer for more information. You can now deallocate the conversation locally.

DEABSVC, deallocate abend services

Facility: SNALU62$

Explanation: An LU services error has occurred, thereby ending the conversation. If the conversation was in receive state when the deallocate was issued, information sent by SNALU62$SEND_DATA is purged. The conversation is in deallocate state.

User Action: Examine subsequent error messages, then see your IBM systems programmer for more information. You can now deallocate the conversation locally.

DEABTIM, deallocate abend timer

Facility: SNALU62$

Explanation: The remote IBM host transaction program has ended (aborted or timed-out) the conversation. If the conversation at the OpenVMS transaction program was in receive state when the deallocate was issued, information sent by SNALU62$SEND_DATA is purged. The conversation is in deallocate state.

User Action: Examine subsequent error messages, then see your IBM systems programmer for more information. You can now deallocate the conversation locally.

DEALNOR, deallocate normal

Facility: SNALU62$

Explanation: The remote IBM host transaction program has ended the conversation under normal circumstances. The conversation is in deallocate state.

User Action: No response required.

EVTFLG, failed to allocate an event flag

Facility: SNALU62$

Explanation: The APPC/LU6.2 Programming Interface could not allocate an event flag.

User Action: Adjust your OpenVMS transaction program code to release all the event flags that it allocates. Event flags are allocated using LIB$GET_EF and deallocated using LIB$FREE_EF.

EXIT, Gateway server task terminated

Facility: SNALU62$

Explanation: The cooperating software in the SNA gateway has failed.

User Action: Look for log messages on the gateway operator's console (console remote for DECSA). (See your DIGITAL system manager or DIGITAL network manager for more information.)

FAIALLBUF, failed to allocate memory for a buffer

Facility: SNA$

Explanation: The APPC/LU6.2 Programming Interface failed to allocate dynamic memory for an internal buffer. The most likely reason is that no free memory is available.

User Action: If you are using class D descriptors, make sure you return used buffers to free memory using LIB$SFREE1_DD or STR$FREE1_DX.

FAIALLCTX, failed to allocate memory for a context block

Facility: SNA$

Explanation: The APPC/LU6.2 Programming Interface failed to allocate memory for an internal context block. The most likely reason is that no free memory is available.

User Action: If you are using class D descriptors, make sure you return used buffers to free memory using LIB$SFREE1_DD or STR$FREE1_DX.

FAIASSCHA, failed to assign a DECnet channel

Facility: SNA$

Explanation: The error indicates an abnormal DECnet condition.

User Action: Examine the subsequent DECnet error messages and report the problem to your system manager.

FAIBLDNCB, failed to build DECnet network connect block

Facility: SNA$

Explanation: The APPC/LU6.2 Programming Interface failed to build a DECnet network connect block in order to communicate with the gateway.

User Action: Examine the subsequent error messages for more information.

FAICONMBX, failed to convert mailbox name

Facility: SNA$

Explanation: The APPC LU6.2 Programming Interface could not create a mailbox for establishing a logical link.

User Action: Examine subsequent error messages to find the reason. The most likely additional message is SYSTEM-F-NOPRIV, which indicates no privilege for attempted operation. This means that you lack TMPMBX privilege.

FAICOPBIN, failed to copy BIND request image into caller's buffer

Facility: SNA$

Explanation: The APPC/LU6.2 Programming Interface could not copy the entire BIND request image into the BIND request buffer provided by the application.

User Action: Make sure you specify a BIND buffer that is large enough to receive the largest BIND the IBM application will send you.

FAIESTLIN, failed to establish a DECnet link to the gateway

Facility: SNA$

Explanation: The APPC/LU6.2 Programming Interface cannot connect to the gateway.

User Action: Examine the subsequent DECnet error messages and take appropriate action.

FATINTERR, internal error in gateway access routines

Facility: SNA$

Explanation: A fatal error has occurred.

User Action: The appearance of this error message implies that the APPC/LU6.2 Programming Interface is operating abnormally. To take corrective action, copy all the messages associated with the fatal error code and take them to your DIGITAL system manager, who will decide what action to take. If you still cannot solve your problem, submit a Software Performance Report (SPR), or call the DIGITAL support hotline, if you have the services.

FMHNOT, FMH data is not supported

Facility: SNALU62$

Explanation: Data records containing function management headers (FMHs) were issued to a remote IBM host transaction program that does not support FM header data. Information sent by SNALU62$SEND_DATA is purged. The conversation is in send state.

User Action: Remove the *fmh data* parameter and rerun the OpenVMS transaction program.

FUNCABORT, access routine function aborted

Facility: SNA$

Explanation: The APPC/LU6.2 Programming Interface procedure did not complete successfully and the session has been or is being terminated.

User Action: Ignore the error. You have or will get notification of an asynchronous event that will tell you why the session has terminated.

FUNNOTVAL, function not valid with port in current state

Facility: SNA$

Explanation: The APPC/LU6.2 Programming Interface is invalid with the port in its current state. You issued APPC/LU6.2 Programming Interface calls in the wrong order — for example, an SNA$TRANSMIT before an SNA$ACCEPT.

User Action: Correct the code in your application.

GATCOMERR, error communicating with gateway node

Facility: SNALU62$, SNA$

Explanation: A fatal communication error has occurred with the gateway node.

User Action: Examine the subsequent DECnet error messages and take appropriate action.

GATINTERR, internal error in gateway node, code *octal_number*, subcode *octal_number*

Facility: SNA$

Explanation: A fatal error has occurred.

User Action: Report the error to your system manager. Also, ensure that the log from the gateway console is saved. It will have messages of the form: GAS — Fatal Session Error FSE$xxx

HOSTERR, error returned from host, *error_text* error message received from host

Facility: SNALU62$

Explanation: The remote LU returned an error message. The error text is a message output by the IBM subsystem that implements the remote LU. CICS is an example of an IBM subsystem.

User Action: See the IBM subsystem documentation for more information on the error text.

ILLASTSTA, ASTs are disabled or an AST routine is currently in progress

Facility: SNA$

Explanation: A call was made to an APPC/LU6.2 Programming Interface procedure either while ASTs were disabled or from within an AST routine.

Because AST delivery is disabled, there is no way that the procedure can complete. Therefore, the procedure has taken no action.

User Action: Change the application so that APPC/LU6.2 Programming Interface procedures are not called from AST routines or with ASTs disabled.

INCVERNUM, Gateway access routines are incompatible with the gateway

Facility: SNA$

Explanation: The software on the gateway is incompatible with the SNA software on the local DECnet node.

User Action: Make sure that the correct versions of the software are installed on both the gateway and the DECnet node.

INSGATRES, insufficient gateway resources for session establishment

Facility: SNA$

Explanation: The gateway has insufficient resources for establishing a session. The active sessions currently in the gateway are using the total resources available.

User Action: Wait until some of the sessions have finished, then retry.

INSRESOUR, insufficient resources to establish session

Facility: SNA$

Explanation: The APPC/LU6.2 Programming Interface could not allocate enough system resources to establish the session.

User Action: Examine the subsequent messages for more information.

INVFILL, RECEIVE_AND_WAIT with FILL(LL) is invalid after FILL(BUFFER)

Facility: SNALU62$

Explanation: You issued an SNALU62$RECEIVE_AND_WAIT procedure with a *fill* type BUFFER following an SNALU62$RECEIVE_AND_WAIT procedure with *fill* type LL. You cannot issue this sequence of procedures without issuing a call to a valid intervening procedure.

User Action: Correct the algorithm in your OpenVMS transaction program and rerun.

INVGDS, invalid GDS ID sent to remote system, code *hex_value*

Facility: SNALU62$

Explanation: The remote LU did not understand a generalized data stream (GDS) ID that was sent by the APPC/LU6.2 Programming Interface on a mapped conversation. This indicates a software problem in the APPC/LU6.2 Programming Interface or the IBM subsystem with which you are communicating.

User Action: Ask your DIGITAL system manager to submit an SPR with a description of the problem, an SNATRACE listing of the affected session, and a transcription of all the error messages.

INVGWYNOD, parameter gwynode is invalid

Facility: SNA$

Explanation: You entered an invalid value in the *gwynode* parameter.

User Action: Examine the call that returned the error and take appropriate action.

INVPAR, invalid value supplied for *parameter_name* parameter

Facility: SNALU62$

Explanation: You entered an invalid value for the *parameter_name* parameter in an APPC/LU6.2 Programming Interface procedure.

User Action: Consult the documentation to determine valid parameter values and correct the OpenVMS transaction program.

INVRECLOG, SNA$DEF_NUMREC is incorrectly defined

Facility: SNA$

Explanation: This internal logical name is set up improperly.

User Action: SNA$DEF_NUMREC is a logical name that determines the number of receives the APPC/LU6.2 Programming Interface keeps outstanding on the DECnet logical link. If you do not wish to use the default value, use the DCL DEFINE command (for example, DEFINE SNA$DEF_NUMREC 5).

INVRES, invalid *resource_id* specified

Facility: SNALU62$

Explanation: You entered an invalid *resource_id* parameter to an APPC/LU6.2 procedure. The *resource_id* for a conversation is returned by the SNALU62$ALLOCATE procedure. This must be supplied on all subsequent procedures issued for this conversation.

User Action: Correct the OpenVMS transaction program to supply the correct *resource_id*.

INVRLST, entry number in *resource_list* is not a valid resource ID

Facility: SNALU62$

Explanation: The *n*th entry in the *resource_list* specified to the SNALU62$WAIT is not a valid resource ID.

User Action: Correct the OpenVMS transaction program to supply a valid list of resource IDs.

INVSES, invalid session_id specified

Facility: SNALU62$

Explanation: You entered an incorrect *session_id* parameter to an APPC/LU6.2 procedure. The *session_id* for a session is returned by either the SNALU62$ACTIVATE_SESSION or SNALU62$GET_ATTRIBUTES procedures.

User Action: Correct the OpenVMS transaction program to supply the correct *session_id*.

LOGUNIDEA, SSCP has deactivated the session

Facility: SNA$

Explanation: The IBM system services control point (SSCP) has deactivated the session by sending a DACTLU command. Some applications deactivate sessions by deactivating the logical unit rather than by sending an UNBIND command.

LUNAMAL, LU_NAME *lu_name* is already defined

Facility: SNALU62$

Explanation: SNALU62$DEFINE_REMOTE was called to define an LU name that already exists.

User Action: Correct the OpenVMS transaction program. Use SNALU62$DEFINE_REMOTE to define an LU name that does not already exist. You can undefine existing LU names with

SNALU62$DELETE by supplying the LU name and specifying no attributes.

LUNAMUN, associated LU_NAME has been undefined

Facility: SNALU62$

Explanation: An SNALU62$GET_ATTRIBUTES procedure was issued for a conversation on an SNA session whose LU name has been undefined.

User Action: Do not undefine the LU name if you want to issue a subsequent SNALU62$GET_ATTRIBUTES procedure.

MAPEFAI, map execution failure

Facility: SNALU62$

Explanation: Reports an error returned by the remote IBM host transaction program or CICS indicating that the LU could not map the data record based on the map name.

User Action: See your IBM transaction programmer and determine why the mapping function failed.

MAPNFOU, map name not found

Facility: SNALU62$

Explanation: Reports an error returned by the remote IBM host transaction program or CICS indicating that the LU could not map the data record because the map name is unknown to the LU.

User Action: See your IBM transaction programmer and determine the correct map name.

MAXSESACT, maximum number of sessions already active

Facility: SNA$

Explanation: You have already established 120 sessions, the maximum number allowed.

User Action: Make sure you have called the disconnect procedure for each session that has terminated.

MUCBALL, failed to allocate an MUCB

Facility: SNALU62$

Explanation: The APPC/LU6.2 Programming Interface failed to allocate memory for an internal data structure (message unit control block). The most likely reason is that no free virtual memory is available.

User Action: If you are using class D descriptors, make sure you return used buffers to free memory using LIB$SFREE1_DD or STR$FREE1_DX. If your process uses a lot of virtual memory, you may need to increase the PGFLQUOTA parameter for that user with the AUTHORIZE utility. See the appropriate OpenVMS System Management Guide for your system for details on how to use this utility.

MUTORCVCHK, MU generated a receive check, sense code *IBM_sense_code*

Facility: SNA$

Explanation: The message unit returned a receive check sense code.

User Action: Consult your IBM manual for the sense code.

MUTOSENDCHK, MU generated a send check, sense code *IBM_sense_code*

Facility: SNA$

Explanation: The message unit returned a send check sense code.

User Action: Consult your IBM manual for the sense code.

NDAVAIL, no data is available

Facility: SNALU62$

Explanation: There is no data available for SNALU62$RECEIVE_ IMMEDIATE.

User Action: Call the SNALU62$RECEIVE_IMMEDIATE* procedure until data is available, or the SNALU62$_OK status code is returned.

NETSHUT, network node is not accepting connects

Facility: SNALU62$

Explanation: The gateway node is not accepting requests to establish a DECnet logical link necessary to support an SNA session. This is probably because the gateway has not finished its initialization sequence.

User Action: Wait a while and try again.

NO_GWYNOD, SNA$DEF_GATEWAY is undefined and *gwynode* was not specified

Facility: SNA$

Explanation: No gateway node was specified and the logical name SNA$DEF_GATEWAY was not defined.

User Action: Either supply an explicit gateway node specification or define SNA$DEF_GATEWAY using the OpenVMS DEFINE command.

NO_SUCACC, access name not recognized by gateway node

Facility: SNA$

Explanation: You specified a nonexistent access name.

User Action: Check with your system manager to determine which access name you need.

NO_SUCPU, PU name not recognized by gateway node

Facility: SNA$

Explanation: Either you or the access name you used specified a nonexistent physical unit.

User Action: Check with your system manager to determine which PU name or access name you need.

NO_SUCSES, session address not recognized by gateway node

Facility: SNA$

Explanation: Either you or the access name you used specified a nonexistent session address.

User Action: Check with your system manager to determine which session address or access name you need.

NOPOST, posting is not active for any of the specified conversations

Facility: SNALU62$

Explanation: SNALU62$WAIT was called when there were no conversations in which posting was active for the SNALU62$WAIT call.

User Action: Remove the call to SNALU62$WAIT and use SNALU62$RECEIVE_AND_WAIT instead, or turn posting on for the conversations that require it, by issuing a SNALU62$POST_ON_RECEIPT verb prior to the call to SNALU62$WAIT.

NOSYNC, conversation was allocated with SYNC_LEVEL(NONE)

Facility: SNALU62$

Explanation: SNALU62$CONFIRM was called for a conversation that was allocated with a synchronization level of NONE.

User Action: Do not issue SNALU62$CONFIRM for conversations whose synchronization level is NONE.

NOTACT, SNA session no longer active

Facility: SNALU62$

Explanation: The SNA session supporting the conversation has been terminated. Subsequent error messages will give more detail.

User Action: Examine the subsequent error messages.

NOTEST, failed to establish SNA session

Facility: SNALU62$

Explanation: The conversation could not be allocated because of failure to establish the SNA session. Subsequent error messages will give more detail.

User Action: Examine the subsequent error messages.

NOTSUP, unsupported value supplied for *parameter_name* parameter

Facility: SNALU62$

Explanation: A value was supplied for a parameter that is not supported by this implementation.

User Action: Supply a supported value for the parameter.

OK, normal successful completion

Facility: SNALU62$

Explanation: Normal completion.

User Action: No response required.

OUTPAR, error writing to *parameter_name* parameter

Facility: SNALU62$

Explanation: A procedure failed to complete because the APPC/LU6.2 Programming Interface could not write into the memory location specified by the APPC/LU6.2 Programming Interface for the parameter.

User Action: Correct the OpenVMS transaction program so that the parameter storage is in writable memory.

PARERR, parameter error, routine name

Facility: SNALU62$

Explanation: A parameter error was detected when the named routine was called. The conversation state remains unchanged. Subsequent error messages will provide more information.

User Action: Examine the subsequent error messages, adjust OpenVMS transaction program logic, then rerun the program.

PARTREC, logical record only partially sent

Facility: SNALU62$

Explanation: An error was returned to SNALU62$CONFIRM because the last SNALU62$SEND issued did not finish with a full record (LL).

User Action: Issue another SNALU62$SEND and specify the rest of the record before you issue the SNALU62$CONFIRM.

PIPNALL, PIP data must not be specified

Facility: SNALU62$

Explanation: The remote LU rejected the conversation allocation request because Program Initialization Parameters (PIP data) were specified on the SNALU62$ALLOCATE procedure. The remote IBM host transaction program has no PIP variables defined.

User Action: Eliminate PIP data on the SNALU62$ALLOCATE procedure. Request that your IBM system manager system generate your host system to accept PIP data.

PLUPROVIO, PLU violated SNA protocol rules, sense code *IBM_sense_code*

Facility: SNA$

Explanation: The primary logical unit violated SNA protocol rules.

User Action: Consult your IBM manual for the sense code.

PRERNTR, program error no truncate

Facility: SNALU62$

Explanation: The remote IBM host transaction program has returned an error message on the SNALU62$RECEIVE_AND_WAIT procedure while the conversation was in send state. A complete record was sent; no truncation occurred. The conversation is in receive state.

User Action: Consult the IBM systems programmer for more information. Also see the *IBM Transaction Programmer's Reference Manual for LU Type 6.2*, Order No. GC30-3084, and the *System Network Architecture Format*

and Protocol Reference Manual: Architecture Logic for LU Type 6.2, Order No. SC30-3269.

PRERPU, program error purging

Facility: SNALU62$

Explanation: The remote IBM host transaction program has returned an error message while the conversation was in receive or confirm state. Information may have been purged. The conversation is now in receive state.

User Action: Consult the IBM systems programmer for more information. Also see the *IBM Transaction Programmer's Reference Manual for LU Type 6.2*, Order No. GC30-3084 and the *System Network Architecture Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*, Order No. SC30-3269.

PRERTR, program error truncate

Facility: SNALU62$

Explanation: The remote IBM host transaction program has returned an error message on the SNALU62$RECEIVE_AND_WAIT procedure while the conversation was in send state. A logical record was truncated. The conversation remains in receive state.

User Action: Consult the IBM systems programmer for more information. Also see the *IBM Transaction Programmer's Reference Manual for LU Type 6.2*, Order No. GC30-3084, and the *System Network Architecture Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*, Order No. SC30-3269.

PROUNBREC, IBM application detected a protocol error, sense code %X'IBM sense code'

Facility: SNA$

Explanation: The IBM application sent an UNBIND request with the indicated sense code. It did this because the application detected the protocol error specified by the code.

User Action: Determine the meaning of the sense code from the IBM documentation and take the appropriate action.

PSWLONG, locally defined session level password is too long

Facility: SNALU62$

Explanation: Error code. The LU-to-LU password can be a maximum of 8 bytes in length.

User Action: Consult your IBM system manager for the correct LU-to-LU password. Correct the password in the SNALU62$DEFINE_REMOTE call.

PSWNDEF, local session level password was not defined

Facility: SNALU62$

Explanation: Error code. Your session level security was defined in the BIND from the remote LU. No LU-to-LU password ID defined.

User Action: Supply a password to the *lu_lu_password* parameter in the SNALU62$DEFINE_REMOTE procedure call.

PUNOTAVA, pu has not been activated

Facility: SNA$

Explanation: BM.

User Action: Ask the VTAM operator to check the line and physical unit (PU) from the IBM host and activate them if necessary. If they are activated, there may be a hardware problem between the gateway and the IBM host.

PUNOTSPE, PU name was not specified

Facility: SNA$

Explanation: In the connect request you did not specify a physical unit, and the access name that you used did not supply one either.

User Action: The circuit name must be either explicitly supplied in the parameter list or implicitly supplied through the access name.

RESFNO, resource failure no retry

Facility: SNALU62$

Explanation: The conversation has been terminated because of a protocol error in the SNA layer. Subsequent messages will identify specific errors. The conversation is in deallocate state.

User Action: Ask your DIGITAL system manager to submit an SPR with a description of the problem, an SNATRACE listing of the affected session, and a transcription of all the error messages.

RESFRET, resource failure retry

Facility: SNALU62$

Explanation: This signifies that the conversation has been temporarily terminated, usually because of a communication failure. The OpenVMS transaction program can retry the conversation. Subsequent messages will provide more information. The conversation is in deallocate state.

User Action: Retry the conversation at a later time.

SESIN_USE, session address is already in use

Facility: SNA$

Explanation: Someone else is using this session address.

User Action: Retry using a different session address. If you are unsure of a valid choice, ask your system manager.

SESNOTAVA, session address has not been activated

Facility: SNA$

Explanation: The SLU has not been activated from the IBM side.

User Action: Ask the IBM VTAM operator to check the logical unit from the IBM host and activate it if necessary.

SESSECF, LU-LU session level security failed

Facility: SNALU62$

Explanation: The remote LU cannot be recognized with the LU-to-LU password defined on the local LU.

User Action: Consult your IBM system manager for the correct LU-to-LU password. Correct the password in the SNALU62$DEFINE_REMOTE call.

SNAASSFAI, failed to assign an I/O channel to _SNA0

Facility: SNA$

Explanation: You did not specify a gateway node name when using the SNA gateway.

User Action: Specify a gateway node name for the gateway you want to use.

STAERR, LU6.2 verb *procedure_name* invalid with conversation in current state

Facility: SNALU62$

Explanation: The OpenVMS transaction program attempted to issue another APPC/LU6.2 Programming Interface procedure while one is in process. The conversation state remains unchanged.

User Action: Adjust OpenVMS transaction program logic and rerun the program.

STRALL, failed to allocate memory for dynamic string

Facility: SNALU62$

Explanation: The APPC/LU6.2 Programming Interface failed to allocate memory for an internal data structure. The most likely reason is that no free virtual memory is available.

User Action: If you are using class D descriptors, make sure you return used buffers to free memory using LIB$SFREE1_DD or STR$FREE1_DX. If your process uses a lot of virtual memory, you may need to increase the PGFLQUOTA parameter for that user with the AUTHORIZE utility. See the appropriate OpenVMS System Management Guide for your system for details about using this utility.

STRCOP, failed to copy string

Facility: SNALU62$

Explanation: The APPC/LU6.2 Programming Interface failed to copy data from one area to another. The most likely reason is that no free virtual memory is available.

User Action: If you are using class D descriptors, make sure you return used buffers to free memory using LIB$SFREE1_DD or STR$FREE1_DX. If your process uses a lot of virtual memory, you may need to increase the PGFLQUOTA parameter for that user with the AUTHORIZE utility. See the appropriate OpenVMS System Management Guide for your system for details about using this utility.

SUBFLD, invalid subfield in BIND Image Structured Data, value *hex value*

Facility: SNALU62$

Explanation: A BIND image was received with an invalid subfield in one of the structured data fields. Valid values for LU6.2 are 0, 2, 3, 4, and 5. This message indicates an SNA protocol problem.

User Action: If the values are 0, 2, 3, 4, or 5, ask your system manager to submit an SPR with a description of the problem, an SNATRACE listing of

the affected session, and a transcription of all the error messages. If the values differ from these, the problem is on the IBM side; see your IBM systems programmer.

SVCENTR, service error no truncate

Facility: SNALU62$

Explanation: The remote IBM host transaction program has returned an error message on the SNALU62$RECEIVE_AND_WAIT procedure while the conversation was in send state. A complete record was sent, no truncation occurred. The conversation is in receive state.

User Action: Consult your IBM systems programmer to determine the reason for the error. Also see the *IBM Transaction Programmer's Reference Manual for LU Type 6.2*, Order No. GC30-3084, and the *System Network Architecture Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*, Order No. SC30-3269.

SVCERPU, service error purging

Facility: SNALU62$

Explanation: The remote IBM host transaction program sent an error message while the conversation was in receive or confirm state. Information may have been purged. The conversation is in receive state.

User Action: Consult your IBM systems programmer to determine the reason for the error. Also see the *IBM Transaction Programmer's Reference Manual for LU Type 6.2*, Order No. GC30-3084, and the *System Network Architecture Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*, Order No. SC30-3269.

SVCERTR, service error truncate

Facility: SNALU62$

Explanation: The remote IBM host transaction program sent an error message on the SNALU62$RECEIVE_AND_WAIT procedure while the conversation was in send state. A logical record was truncated. The conversation remains in receive state.

User Action: Consult your IBM systems programmer to determine the reason for the error. Also see the *IBM Transaction Programmer's Reference Manual for LU Type 6.2*, Order No. GC30-3084, and the *System Network Architecture Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*, Order No. SC30-3269.

SYNNSUP, requested sync level not supported by remote program

Facility: SNALU62$

Explanation: The remote LU rejected the conversation allocation request because the remote IBM host transaction program does not support the synchronization level specified in the SNALU62$ALLOCATE procedure.

User Action: Specify a synchronization level that is supported by the remote IBM host transaction program.

TPNALDF, TP_NAME tp_name is already defined

Facility: SNALU62$

Explanation: SNALU62$DEFINE_TP was called to define a TP name that already exists.

User Action: Correct the OpenVMS transaction program. Use SNALU62$DEFINE_TP to define a TP name that does not already exist. You can undefine existing TP names with SNALU62$DELETE by supplying the TP name and specifying no attributes.

TPNLON, TPN is too long

Facility: SNALU62$

Explanation: The transaction program name (TPN) specified on the SNALU62$ALLOCATE procedure is too long. A TPN name must be no more than 64 characters in length.

User Action: Correct the OpenVMS transaction program to specify a TPN name of fewer than 64 characters.

TPNNREC, TPN not recognized by remote system

Facility: SNALU62$

Explanation: The remote LU rejected the conversation allocation request because the transaction program name (TPN) specified on the SNA$ALLOCATE procedure was not recognized by the remote LU.

User Action: Specify the correct TPN or ask your IBM systems programmer to configure the remote IBM host transaction program correctly on the IBM system.

TPNNRET, remote TPN is not available, no retry possible

Facility: SNALU62$

Explanation: The conversation allocation request was rejected because the remote LU could not start the requested transaction program (TPN). This condition must be corrected before you retry.

User Action: See the IBM systems programmer to determine the reason for the problem.

TPNRET, remote TPN is not available, retry later

Facility: SNALU62$

Explanation: The conversation allocation request was rejected because the remote LU could not start the requested remote IBM host transaction program (TPN). This condition is transient.

User Action: Reissue the conversation allocation request at a later time.

UNABD0, unacceptable BIND image, byte number field name *field_name*

Facility: SNALU62$

Explanation: An SNA BIND image was received with the indicated invalid field. Either you are using an LU that is not defined on the IBM system for an LU6.2 type session or the LU has been incorrectly configured.

User Action: Have your IBM systems programmer define the LU for LU6.2 use. See the *DECnet SNA Gateway-CT Guide to IBM Parameters* or the *DECnet SNA Gateway-ST Guide to IBM Parameters* for more details.

UNABD1, unacceptable BIND image, byte number, bit *number*, field name *field_name*

Facility: SNALU62$

Explanation: An SNA BIND image was received with the indicated invalid field. Either you are using an LU that is not defined on the IBM system for an LU6.2 type session or the LU has been incorrectly configured. Note that the bit numbers are in IBM notation. That is, the most significant bit in a byte is bit 0; the least significant is bit 7.

User Action: Have your IBM systems programmer define the LU for LU6.2 use. See the *DECnet SNA Gateway-CT Guide to IBM Parameters* or the *DECnet SNA Gateway-ST Guide to IBM Parameters* for more details.

UNABD2, unacceptable BIND image, byte number, bits *number*, field name *field_name*

Facility: SNALU62$

Explanation: An SNA BIND image was received with indicated invalid field. Either you are using an LU that is not defined on the IBM system for an LU6.2 type session or the LU has been incorrectly configured. Note that the bit numbers are in IBM notation. That is, the most significant bit in a byte is bit 0; the least significant is bit 7.

User Action: Have your IBM systems programmer define the LU for LU6.2 use. See the *DECnet SNA Gateway-CT Guide to IBM Parameters*, or the *DECnet SNA Gateway-CT Guide to IBM Parameters* for more details.

UNABLELUCB, unable to obtain lucb

Facility: SNA$

Explanation: Insufficient virtual memory.

User Action: Increase virtual memory.

UNABLEMUCB, unable to obtain mucb

Facility: SNA$

Explanation: Insufficient virtual memory.

User Action: Increase virtual memory.

UNABLESCB, unable to obtain scb

Facility: SNA$

Explanation: Insufficient virtual memory.

User Action: Increase virtual memory.

UNARAN, needed a value in range *hex_value* to *hex_value*, received *hex_value*

Facility: SNALU62$

Explanation: A value was received in an SNA BIND image that was out of range. The particular BIND image field in error is reported in a preceding message. Either you are using an LU that is not defined on the IBM system for an LU6.2 type session or the LU has been incorrectly configured.

User Action: Have your IBM systems programmer define the LU for LU6.2 use. See the *DECnet SNA Gateway-CT Guide to IBM Parameters*, or the *DECnet SNA Gateway-ST Guide to IBM Parameters* for more details.

UNAVAL, expected *hex_value*, received *hex_value*

Facility: SNALU62$

Explanation: An invalid value was received in an SNA BIND image. The particular BIND image field in error is reported in a preceding message. Either you are using an LU that is not defined on the IBM system for an LU6.2 type session or the LU has been incorrectly configured.

User Action: Have your IBM systems programmer define the LU for LU6.2 use. See the *DIGITAL SNA Gateway Guide to IBM Parameters* for more details.

UNBINDREC, UNBIND request received from IBM application

Facility: SNA$

Explanation: The IBM application has terminated the session by sending a normal UNBIND RU.

User Action: Determine the meaning from the IBM documentation and take the appropriate action.

UNDTPN, TP_NAME *tp_name* is not defined

Facility: SNALU62$

Explanation: SNALU62$DELETE was called to undefine a TP name that does not exist.

User Action: Correct the OpenVMS transaction program that deletes a TP name that does exist.

UNLUNAM, LU_NAME *lu_name* is not defined

Facility: SNALU62$

Explanation: SNALU62$DELETE was called to undefine an LU name that does not exist.

User Action: Correct the OpenVMS transaction program that deletes an LU name that does exist.

UNSUC, verb did not execute successfully

Facility: SNALU62$

Explanation: The APPC/LU6.2 procedure did not execute successfully. The conversation state remains unchanged. Subsequent messages will provide more detail.

User Action: Correct errors pointed out in subsequent error messages.

UNUUNBREC, UNBIND of type *hex_type* received from IBM application

Facility: SNA$

Explanation: The IBM application sent the specified type of UNBIND request.

User Action: Determine the meaning of this from the IBM documentation on the UNBIND request and take the appropriate action.

USRNACC, user does not have access to the requested resource

Facility: SNALU62$

Explanation: The remote LU denies the user access to the transaction program. The access security information is invalid.

User Action: See your IBM system manager for access to the specified transaction program.

I

Problem Solving

This appendix offers solutions to problems you may encounter when using the APPC/LU6.2 Programming Interface. The following table lists some common problems. Each problem and its solution then begin a new page.

Problem Number	Symptom
1	You get compile time errors with your APPC/LU6.2 Programming Interface transaction program.
2	The linker indicates that the APPC/LU6.2 Programming Interface symbols in your program are not defined.
3	You receive an error message stating that a particular parameter is invalid.
4	You receive an error message stating that the contents of a particular parameter are faulty.
5	You successfully established a session but you cannot request the required CICS transaction program.
6	You receive the SNALU62$_STAERR error code from APPC/LU6.2 Programming Interface procedures.
7	You receive a BUGCHECK error while your application is running.
8	Your application hangs; that is, it enters either the LEF or MWAIT state.
9	You receive the secondary error code SNALU62$_HOSTERR followed by an IBM error code.

Problem 1

You get compile time errors with your APPC/LU6.2 Programming Interface transaction program.

Solution:

This problem could result from one of the following syntax errors:

- Spelling errors in your code
- Incorrect use of definitions

  Make sure you include one of the following definition files from the directory SYS$LIBRARY:

  SNALU62DF.ADA
  SNALU62DF.BAS
  SNALU62DF.FOR
  SNALU62DF.H
  SNALU62DF.LIB
  SNALU62DF.MAR
  SNALU62DF.PAS
  SNALU62DF.PEN
  SNALU62DF.PLI
  SNALU62DF.R32

  The method for including the definition file is language dependent. For the exact syntax required, see the programmer's guide for the language that you are using.

Problem 2

The linker indicates that the APPC/LU6.2 Programming Interface symbols in your program are not defined.

Solution:

Make sure that you have linked to the APPC/LU6.2 Programming Interface shareable image section. The following command sequence produces the proper results:

```
$ LINK/MAP test,SYS$INPUT/OPTIONS  Return
SYS$SHARE:SNALU62SH/SHARE  Return
Ctrl/Z
$
```

where *test* is the name of your program.

Problem 3

You receive an error message stating that a particular parameter is invalid.

Solution:

An invalid parameter error message may indicate that you supplied a parameter in an incorrect form. This could result, for example, from passing the parameter by descriptor when you meant to pass it by reference. Check your code.

Problem 4

You receive an error message stating that the contents of a particular parameter are faulty.

Solution:

See the explanation of the particular error.

Problem 5

You successfully established a session but you cannot request the required CICS transaction program.

Solution:

Check that you have translated the transaction program name (TPN) to EBCDIC.

Problem 6

You receive the SNALU62$_STAERR error code from APPC/LU6.2 Programming Interface procedures.

Solution:

Add code to the OpenVMS transaction program to trace the procedures called, the status returned and, where applicable, the values of the *what_received* and *rts_rec* parameters. Use the information in the state transition sections accompanying each procedure call in Chapter 4 to determine the conversation state after each call. This tells you what the conversation state was before the procedure that failed. You can now determine the permissible conversation states for the procedure you want to call by using the table in Appendix D. Correct your program logic so that the correct state matches the procedure you want to call.

For example, in the following FORTRAN segment, the SNALU62$DEALLOCATE procedure call fails with the SNALU62$_STAERR error code:

```
STATUS = SNALU62$ALLOCATE (...)
IF (.NOT. STATUS) THEN ...
.
.
.
ENDIF
STATUS = SNALU62$DEALLOCATE (RESOURCE_ID,
STATUS_VECTOR,
%REF(SNALU62$K_LOCAL))
IF (.NOT. STATUS) THEN ...
.
.
.
ENDIF
```

Chapter 4 shows that the SNALU62$ALLOCATE procedure enters the send state upon successful completion; thus, the conversation is in the send state when the SNALU62$DEALLOCATE procedure is called. Looking at Appendix D you can see that the SNALU62$DEALLOCATE procedure with the *type* parameter set to SNALU62$K_LOCAL requires that the conversation be in the deallocate state. In other words, the OpenVMS transaction program cannot deallocate the conversation locally because the conversation is in the send state. To enter the deallocate state, the remote IBM transaction procedure must issue the deallocate request. If the OpenVMS transaction program wants to deallocate the conversation at that point, it must set the *type* parameter to one of the following:

SNALU62$K_SYNC_LEVEL

SNALU62$K_FLUSH
SNALU62$K_ABEND_PROG
SNALU62$K_ABEND_SVC
SNALU62$K_ABEND_TIMER

Problem 7

You receive a BUGCHECK error while your application is running.

Solution:

Check to see that:

- Your application is not writing into undeclared or unallocated memory.
- You are using dynamic memory correctly; that is, allocating and deallocating memory correctly. If you have deallocated a piece of memory, be sure that you are not still writing to it.

If you are not using dynamic memory, report the problem with a Software Performance Report (SPR). For information on how to report your problem, refer to the *DECnet SNA Gateway-ST Problem Solving (OpenVMS)*, *DECnet SNA Gateway-CT Problem Solving (OpenVMS & ULTRIX)*, or *OpenVMS SNA Problem Solving*

Problem 8

Your application hangs; that is, it enters either the LEF or MWAIT state.

Solution:

Check the size of your process quotas, particularly the ASTLM quota. Your quotas may be too small.

Problem 9

You receive the secondary error code SNALU62$_HOSTERR followed by an IBM error code.

For example, you receive the following:

```
    %SNALU62-E-DEABPR, deallocate abend program
    -SNALU62-E-HOSTERR, error returned from host
DFH2206I TRANSACTION AIBR ABEND ATCV . BACKOUT SUCCESSFUL
```

Solution:

The secondary SNALU62$_HOSTERR code issued by the APPC/LU6.2 Programmming Interface indicates that a problem has occurred in the remote IBM host transaction program. The DFH error code (IBM code) provides details about the problem the remote transaction program is having. In this example, the problem may be the result of a lack of cooperation between the OpenVMS and the remote IBM host transaction programs.

Look up the IBM error code in the *IBM CICS/VS Version 1, Release 7, Application Programmer's Reference Manual (Command Level)*, IBM Order No. SC33-0241, to determine what the problem is. Then use the OpenVMS Symbolic Debugger to determine at what point in your OpenVMS transaction program the problem occurs. Correct the problem, then recompile and link your program. For information about the Symbolic Debugger, see the *OpenVMS Debugger Manual*, previously entitled, *OpenVMS Debugger Reference Manual*.

J

DIGITAL SNA Access Server for Windows NT Programming Considerations

This appendix lists the APPC interface differences you must consider when migrating to the DIGITAL SNA Access Server for Windows NT product. For more information about any of these issues, see the appropriate verb definition and the introductory information in the first three chapters.

J.1 SNALU62$DEFINE_REMOTE

- The *plu* parameter must reference a remote LU alias on the Microsoft SNA Server.
- The *logon* parameter must reference a mode name on the Microsoft SNA Server.
- The DIGITAL SNA Access Server for Windows NT does not support the *user_data* parameter.
- The *pu* parameter, and optionally the *sesaddr* parameter, specify the local LU used on the Microsoft SNA Server. These parameters are used to create an old-style LU name that must be defined in the Name Mapping section of the Access Server's configuration. In addition, the name mapping must reference an LU6.2 LU in the Microsoft SNA Server.
- The use of the *gwynode* parameter is expanded to include the ability to reference a OpenVMS logical that defines the LU6.2 Server on the DIGITAL SNA Access Server for Windows NT.

Note

If you choose to use the DECnet transport between the APPC interface and the DIGITAL SNA Access Server for Windows NT, you must have DIGITAL PATHWORKS software installed on the DIGITAL SNA Access Server for Windows NT system.)

- Errors in the *plu*, *logon*, and *pu* parameters might not be reported until another verb, for example, the SNALU62$ACTIVATE_SESSION verb, references the parameters.
- The access name specified in the *accname* parameter must reference an access name on the Access Server that references only LU6.2 LUs. On the older DECnet SNA Gateways, an access name could reference both LUA LUs and LU6.2 LUs. This is not supported on the Access Server.

  On the older DECnet SNA Gateways, the PU and the session address were completely independent. That is, an access name could contain a PU (for example, SNA-0) and a session list (for example, 1 - 10) resulting in 10 LUs (SNA-0.1 through SNA-0.10). An APPC program could specify another PU to override the access name's PU (for example, SNA-1) but still use the access name's session list resulting in 10 different LUs (SNA-1.1 through SNA-1.10) that shared the same session address component.

  Using the Access Server, the PU and session address are dependent. Unlike the older DECnet SNA Gateways, when an APPC program specifies another PU to override the access name's PU (for example, SNA-1 overriding SNA-0), the LUs referenced use whatever session addresses were defined for the new PU. For example, if SNA-1 had sessions 51 through 60 defined, using SNA-1 would reference LUs SNA-1.51 through SNA-1.60)

J.2 SNALU62_DEFINE_TP

- The TP name must match the standard Microsoft translation table. If a custom translation is used, the characters must be contained in the Microsoft G translation table. See the Microsoft documentation for more information about Microsoft translation tables.

J.3 SNALU62$ALLOCATE

- The value for the *mode_name* parameter must match a mode name defined on the DIGITAL SNA Access Server for Windows NT.
- The *profile* parameter is not supported.

J.4 SNALU62$DEALLOCATE

- On mapped conversations, all three abend types for the *type* parameter are changed to the generic ABEND option supported by the Microsoft SNA Server. The actual ABEND type returned to the remote transaction program is indeterminate.

J.5 SNALU62$GET_ATTRIBUTES

- The DIGITAL SNA Access Server for Windows NT does not return any value for the *security_profile* parameter.

Glossary

ABEND

Certain errors related to the execution of procedures can cause an abnormal ending (ABEND) of the transaction program. When the LU terminates a program because of an ABEND condition, it deallocates all active conversations.

advanced program-to-program communication (APPC)

IBM's generic interface that uses the LU6.2 architecture. Eventually, LU6.2 will be the common LU for all IBM products. LU6.2 provides a common language (what IBM refers to as verbs and DIGITAL refers to as procedures) that transaction programs call as subroutines to perform communication functions. In advanced program-to-program communication, two cooperating transaction programs communicate over the network in a peer-to-peer fashion.

APPC/LU6.2 Programming Interface

Shorthand for the DIGITAL SNA APPC/LU6.2 Programming Interface for OpenVMS software that performs the LU6.2 functions on behalf of the OpenVMS user.

AST

Asynchronous System Trap (AST) services are documented in the *OpenVMS System Services Reference Manual.*

basic conversations

Conversations that require the transaction to build and interpret generalized data stream (GDS) headers when they are used. The transaction is responsible for error recovery and data-stream mapping. (See also **conversation**).

CICS

See **Customer Information Control System**.

conversation

A short-term logical connection between two transaction programs that allows a synchronous exchange of messages during a session. A conversation can be of whatever duration the transaction program requires.

Customer Information Control System (CICS)

An IBM application subsystem that provides basic functions for data communications on an IBM host.

descriptor

A data structure used in a calling procedure for passing argument types, addresses, and other information.

distributed transaction processing

Distributed processing between transaction programs that cooperate with one another.

Domain Gateway

Shorthand for DIGITAL SNA Domain Gateway. The DIGITAL SNA Domain Gateway is a multiple system connection into a IBM SNA network. It is a system of hardware and software that handles the protcol differences between the IBM SNA network and the Digital Equipment Corportation DIGITAL network.

event flag

Event flags are status posting bits maintained by OpenVMS for general programming use. In the APPC/LU6.2 Programming Interface, an event is set to indicate asynchronous completion of a procedure.

function management header (FMH)

Control information that allows an LU to transmit a data stream to a specific destination and control the presentation of the data at that destination. FMHs are the means by which an LU selects the functions it wants the presentation services components of its session partner to perform. For more information about FMHs, see IBM's *Systems Network Architecture — Sessions Between Logical Units*, Order No. GC20-1868.

Gateway

Shorthand for the DIGITAL SNA Gateway. The DIGITAL SNA Gateway is a multiple system connection into an IBM SNA network. It is a system of hardware and software that handles the protocol differences between the IBM SNA network and the Digital Equipment Corporation DIGITAL network.

GDS

See **generalized data stream**.

generalized data stream (GDS)

A standard data stream used in LU6.2 communication. (See also **basic conversations** and **mapped conversations**.)

I/O status vector

A mechanism that provides the OpenVMS transaction program with complete information about error conditions. The status vector provides a top-level completion code, and in many cases, further qualifying codes.

Intersystem Communication (ISC)

CICS method of enabling communication between separate systems by means of programmable interfaces and SNA. (See also **CICS**.)

ISC

See **Intersystem Communication**.

logical link

A temporary conversation path established between two transaction programs in a DIGITAL network.

logical unit (LU)

An interface through which a user (transaction program) gains access to the SNA network. The LU provides an environment (that is, provides services for the transaction program so it can use the network) in which transaction programs can execute.

LU

See **logical unit**.

mapped conversations

Conversations that take place between user-written transaction programs. Mapped conversations transform all data being sent to another transaction into the generalized data stream (GDS) and then restore the data to its original form before the destination program receives it. The OpenVMS transaction program is not aware of the GDS headers and is not responsible for using or interpreting them.

OpenVMS SNA

VMS SNA is a single-system connection to an IBM SNA network, supported by OpenVMS VAX Version 6.1 and Version 6.2 only.

OpenVMS transaction program

The OpenVMS application the user writes using this product.

Peer Server

Shorthand for DIGITAL SNA Peer Server. The DIGITAL SNA Peer Server is a multiple system connection into an IBM SNA network. It is a system of hardware and software that handles the protocol differences between the IBM SNA network and the Digital Equipment Corporation DIGITAL network.

physical unit

An interface through which a user (transaction program) gains access to the services of the user's node resources.

PIP

See **program initialization parameter**.

procedure

An OpenVMS routine entered by means of a call in the transaction program.

program initialization parameter (PIP)

The means of passing program initialization parameters to the destination transaction program.

reference

An argument in an APPC/LU6.2 Programming Interface procedure passed by address and usually expressed as a reference name or label associated with an area or field.

remote IBM host transaction program

The transaction program residing on the IBM system with which the OpenVMS transaction program communicates.

session

A logical connection that permits communication between two logical units.

state

The condition of a conversation at a particular point in time. For example, when a conversation is in receive state, the transaction program cannot send data; it can only receive data. The state of a conversation determines what procedures the OpenVMS transaction program can call.

TPN

See **transaction program name**.

transaction programs

Programs that have been designed to communicate with one another. A transaction usually involves specific input data that causes the execution of a specific task or job.

transaction program name (TPN)

The identifier of the remote IBM host transaction program that the OpenVMS transaction program wants to engage in an LU6.2 conversation.

Virtual Telecommunications Access Method (VTAM)

Software residing in the host IBM system that controls data communications across SNA networks.

Index

O

P

R

S

T

U