# Digital SNA Application Programming Interface for OpenVMS

## Programming

Part Number: AA-P591G-TE

**May 1996**

This manual supplies information about the services provided by the Digital SNA Application Programming Interface that enables 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 6.1, 6.2, or 7.0
OpenVMS Alpha Versions 6.1, 6.2, or 7.0

**Software Version:** Digital SNA Application Programming Interface for OpenVMS, Version 2.4

**May 1996**

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation or EDS. Digital Equipment Corporation or EDS assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

Digital conducts its business in a manner that conserves the environment.

Restricted Rights: Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

Copyright © 1989, 1996 Digital Equipment Corporation, EDS Defense Limited
All Rights Reserved.

The following are trademarks of Digital Equipment Corporation: Alpha, DEC, DEC/CMS, DEC /MSS, DECnet, DECsystem-10, DECSYSTEM-20, DECUS, DECwriter, DIBOL, EduSystem, IAS, MASSBUS, OpenVMS, PDP, PDT, RSTS, RSX, UNIBUS, VAX, VAXcluster, VMS, VT, and the Digital logo.

IBM is a registered trademark of International Business Machines Corporation.

# Contents

**Preface** . . . . . . . . . . vii

## 1 Introduction

1.1 API Features . . . . . . . . . . 1–2
1.2 IBM and SNA Concepts . . . . . . . . . . 1–3
1.3 Common Interface Applications . . . . . . . . . . 1–3

## 2 Concepts and Terms

2.1 What is an LU-LU Type 0 Session? . . . . . . . . . . 2–1
2.2 Establishing an LU-LU Session . . . . . . . . . . 2–2
2.2.1 Issuing an Active Connect Request . . . . . . . . . . 2–2
2.2.2 Issuing a Passive Connect Request . . . . . . . . . . 2–4
2.2.3 Accepting or Rejecting a BIND Request . . . . . . . . . . 2–6
2.3 Using Request/Response Units . . . . . . . . . . 2–7
2.4 Sending Request Units . . . . . . . . . . 2–7
2.4.1 The Request Header . . . . . . . . . . 2–7
2.4.1.1 Chaining Indicators . . . . . . . . . . 2–8
2.4.1.2 Change Direction Indicator . . . . . . . . . . 2–8
2.4.1.3 Bracketing Indicators . . . . . . . . . . 2–8
2.4.1.4 Response Indicators . . . . . . . . . . 2–9
2.4.2 Sequence Numbers and Unique Identifiers . . . . . . . . . . 2–9
2.5 Transmitting User Data in Request Unit Chains . . . . . . . . . . 2–10
2.6 Transmitting Control Information in Request Unit Chains . . . . . 2–13
2.7 Transmitting Response Units . . . . . . . . . . 2–14
2.8 Receiving Request/Response Units . . . . . . . . . . 2–15
2.9 Terminating a Session . . . . . . . . . . 2–18
2.10 Reestablishing a Session . . . . . . . . . . 2–18

## 3 API Features


3.1 Status Codes . . . 3–1
3.1.1 Function Value Returns . . . 3–1
3.1.2 The I/O Status Vector . . . 3–2
3.2 Synchronous and Asynchronous Operation . . . 3–5
3.2.1 Synchronous Mode . . . 3–5
3.2.2 Asynchronous Mode . . . 3–6
3.3 Supplying Access Information to the IBM Host . . . 3–8
3.4 Message Classes and Types . . . 3–9
3.5 State Machine Information . . . 3–11
3.6 Asynchronous Event Notification . . . 3–12


## 4 SNA Functions


4.1 The LU Services Layer . . . 4–1
4.1.1 Network Services . . . 4–1
4.1.2 Presentation Services . . . 4–2
4.2 The Data Flow Control Layer . . . 4–3
4.2.1 Function Management Profiles . . . 4–3
4.2.2 Send/Receive Modes . . . 4–3
4.2.2.1 Half Duplex Flip-Flop Mode . . . 4–3
4.2.2.2 Half Duplex Contention Mode . . . 4–4
4.2.2.3 Duplex Mode . . . 4–4
4.2.3 Chains . . . 4–4
4.2.4 Response Types . . . 4–5
4.2.4.1 No Response Chains . . . 4–5
4.2.4.2 Exception Response Chains . . . 4–5
4.2.4.3 Definite Response Chains . . . 4–5
4.2.5 Request/Response Mode Protocols . . . 4–6
4.2.5.1 Request Modes . . . 4–6
4.2.5.2 Response Modes . . . 4–6
4.2.6 Brackets . . . 4–7
4.2.7 Data Flow Control Requests . . . 4–8
4.2.7.1 Cancel Request . . . 4–8
4.2.7.2 Pause Requests . . . 4–9
4.2.7.3 Cleanup Requests . . . 4–9
4.2.7.4 Signal Request . . . 4–10
4.2.7.5 Bracketing Requests . . . 4–10
4.2.7.6 LU Status Request . . . 4–10
4.3 Transmission Control Layer . . . 4–11

4.3.1 Connection Point Manager . . . . . 4–11
4.3.1.1 Building the Request/Response Header . . . . . 4–11
4.3.1.2 Assigning Sequence Numbers and Unique Identifiers . . . 4–11
4.3.1.3 Pacing . . . . . 4–12
4.3.2 Session Control . . . . . 4–12
4.3.2.1 BIND and UNBIND Requests . . . . . 4–13
4.3.2.2 Starting and Clearing Data Traffic . . . . . 4–13
4.3.2.3 Recovery/Resynchronization Functions . . . . . 4–14

5 Procedure Calling Formats

5.1 SNALU0$EXAMINE_STATE . . . . . 5–2
5.2 SNALU0$RECEIVE_MESSAGE . . . . . 5–4
5.3 SNALU0$REQUEST_CONNECT . . . . . 5–8
5.4 SNALU0$REQUEST_RECONNECT . . . . . 5–11
5.5 SNALU0$REQUEST_DISCONNECT . . . . . 5–12
5.6 SNALU0$TRANSMIT_MESSAGE . . . . . 5–13
5.7 SNALU0$TRANSMIT_RESPONSE . . . . . 5–17

6 Compiling and Linking a Transaction Program

6.1 Creating and Compiling Your Program . . . . . 6–1
6.2 Linking Your Program to the Shareable Program Image . . . . . 6–2

A Summary Chart of Procedure Parameter Notation

B BIND Request Parameters

C The Request Response Header

D Definitions for the Application Programming Interface

E Programming Examples

E.1 FORTRAN Programming Example . . . . . E–1
E.2 FORTRAN Definition Files . . . . . E–14
E.3 COBOL Programming Example . . . . . E–15
E.4 MACRO Programming Example . . . . . E–24
E.5 VAX PL/I Programming Example . . . . . E–40
E.6 Pascal Programming Example . . . . . E–48
E.7 Pascal Symbol and Structure Definitions . . . . . E–58

E.8 C Programming Example . . . . . . . . . . E–59

## F Status Codes

## G Correlation of Procedures and Status Messages for the API

## Index

## Figures

1–1 Digital SNA Network . . . . . . . . . . 1–4
2–1 An Active Connect Request . . . . . . . . . . 2–4
3–1 Status Vector . . . . . . . . . . 3–4
4–1 SNA Layers . . . . . . . . . . 4–2
C–1 Request/Response Header . . . . . . . . . . C–3
G–1 Correlation of Procedures and Status Messages for the API . . . . . . . . . . G–2

## Tables

3–1 Request Unit Classes and Types . . . . . . . . . . 3–9
3–2 Symbols Returned by the SNALU0$EXAMINE_STATE Procedure . . . . . . . . . . 3–11
B–1 Symbolic Codes for the BIND Request . . . . . . . . . . B–1
B–2 A BIND Request . . . . . . . . . . B–5
C–1 Symbolic Codes for the Request/Response Header . . . . . . . . . . C–1
C–2 Request Header . . . . . . . . . . C–4
C–3 Response Header . . . . . . . . . . C–7
D–1 Definitions for the API . . . . . . . . . . D–1

# Preface

The Digital SNA Application Programming Interface (API) for OpenVMS is a Digital Equipment Corporation software product. It enables OpenVMS users to communicate with remote IBM host software programs on systems running OpenVMS SNA and connected to either of the following SNA Gateways:

- DECnet SNA Gateway-ST
- DECnet SNA Gateway-CT
- Digital SNA Domain Gateway-CT
- Digital SNA Domain Gateway-ST
- Digital SNA Peer Server

The API allows you to develop applications on an OpenVMS system that require support for an IBM SNA logical unit (LU) session for LU type 0, 1, 2, or 3.

**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, or the OpenVMS SNA (OpenVMS VAX Version 6.1 and Version 6.2 only) when used in this manual.

---

## Manual Objectives

The *Digital SNA Application Programming Interface for OpenVMS Programming* manual provides the information you need to write an application on an OpenVMS system to establish an LU-LU session with a program residing in an IBM host.

## Intended Audience

This manual is designed for OpenVMS VAX and OpenVMS Alpha programmers. To use the API, you must know IBM's Systems Network Architecture (SNA). You must also know about the application subsystem you will connect to on the IBM host, and about the protocols for the LU types you plan to implement using the API.

## Changes and New Features

The Digital SNA Application Programming Interface (API) for OpenVMS, Version 2.4 differs from the Version 2.3 product only in that it includes support for utilizing TCP/IP to communicate between API and the SNA Gateways (Domain and/or Peer Server).

The information relevant to TCP/IP transport support include:

- SNA_TCP_PORT logical
- SNA_TRANSPORT_ORDER logical
- Specifying TCP/IP hostnames

### SNA_TCP_PORT Logical

The SNA_TCP_PORT logical refers to the remote connection TCP/IP port. The default connection TCP/IP port number is 108. For example, if you want the remote connection TCP/IP port number to be 1234, you can enter the following command line:

```
$ define SNA_TCP_PORT 1234
```

If you want the remote connection TCP/IP port to be made to a service defined and enabled in the UCX database; for example *service_name*, you can enter the following command line:

```
$ define SNA_TCP_PORT service_name
```

### SNA_TRANSPORT_ORDER Logical

The SNA_TRANSPORT_ORDER logical refers to a transport list, which is used in automatic selection of transports. Connections are attempted once for each transport in the list until either a successful connection is made, or an error is returned when all transports in the list fail to connect.

For example, if you want the software to try the DECnet transport and if this fails then to try the TCP/IP transport, you can enter the following command line:

```
$ define SNA_TRANSPORT_ORDER "decnet, tcp"
```

If you want the software to try the TCP/IP transport and if this fails then to try the DECnet transport, you can enter the following command line:

```
$ define SNA_TRANSPORT_ORDER "tcp, decnet"
```

If you want the software to never try the DECnet transport and to try only the TCP/IP transport, you can enter the following command line:

```
$ define SNA_TRANSPORT_ORDER "nodecnet, tcp"
```

If you want the software to never try the TCP/IP transport and to try only the DECnet transport, you can enter the following command line:

```
$ define SNA_TRANSPORT_ORDER "decnet, notcp"
```

**Note**

If the SNA_TRANSPORT_ORDER logical is not defined, the default transport order for OpenVMS Alpha will be DECnet, TCP/IP; and the default transport order for OpenVMS VAX will be local, DECnet, TCP/IP.

## Specifying TCP/IP Hostnames

If you want to specify a full path hostname, the hostname must be enclosed in a pair of double-quotes; for example, "foo.bar.company.com".

If you want the TCP/IP transport to be used as the preferred transport, without specifying a TCP/IP full path hostname, then define the SNA_TRANSPORT_ORDER with "tcp" as the first element in the transport list.

If the hostname ends with a single full-colon (":"), then the TCP/IP transport will be used; for example, "foo:" or foo:.

**Note**

If you specify a double full-colon ("::"), you force the DECnet transport to be used; for example, "foo::" or foo::.

## Structure of This Manual

This manual consists of six chapters and eight appendixes.

| | |
|---|---|
| Chapter 1 | Discusses the Digital SNA application interface products and the features of the API. |
| Chapter 2 | Provides an overview of the API and how your OpenVMS application can make calls to it. |
| Chapter 3 | Describes the features of the API that help you write and execute your application. |
| Chapter 4 | Provides an overview of the SNA functions that the OpenVMS application is required to provide in order to communicate with an IBM application. |
| Chapter 5 | Presents the calling format and parameter list for each procedure provided by the API. |
| Chapter 6 | Describes the procedure for linking an OpenVMS application to the API by means of a shareable image. |
| Appendix A | Provides a summary of the notation used to describe parameters in the API. |
| Appendix B | Provides locations, values, and meanings for the BIND request parameters. |
| Appendix C | Provides locations, values, and functions for the request/response header. |
| Appendix D | Provides symbols, values, and meanings to use when you write your application if a definition file is not supplied for the language you want to use. |
| Appendix E | Provides programming examples in various languages with accompanying explanatory text. |
| Appendix F | Describes the status codes that the API returns to the OpenVMS application. |
| Appendix G | Correlates procedures and status messages used by the API. |

## Associated Documents

The following is a list of documents related to the Application Programming Interface:

- *Digital SNA Application Programming Interface for OpenVMS Installation*
- *Digital SNA Application Programming Interface for OpenVMS Problem Solving*

- *Digital SNA Application Programming Interface for OpenVMS Programming*

You should have the following Digital documents available for reference when you use the Application Programming Interface:

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

You may need to refer to one of the following IBM documents:

- *CICS/OS/VS IBM 3790/3730/8100 Guide*, Order No. SC33-0159
- *CICS/VS Version 1 Release 7 IBM 3790/3730 Guide*, Order No. SC33-0075
- *IMS/VS Version 1 Programming Guide for Remote SNA Systems*, Order No. SH20-9054

- *Systems Network Architecture Format and Protocol Reference Manual: Architectural Logic*, Order No. SC30-3112
- *Systems Network Architecture Formats*, Order No. GA27-3136

## Conventions Used in This Manual

This manual uses the following conventions:

| Convention | Meaning |
|---|---|
| `special type` | This special type indicates an example of user input. |
| UPPERCASE | Uppercase letters in command syntax indicates keywords that you can enter. You can enter keywords in either uppercase or lowercase. |
| *italics* | Represent variables for which you must supply a value. |
| [ ] | Square brackets in command syntax statements indicate that the enclosed value(s) are optional. Default values apply for unspecified options. (Do not type the brackets.) |
| { } | Braces in command syntax statements indicate that you must specify one, and only one, of the enclosed values. (Do not type the braces.) |
| ( ) | Parentheses enclose a group of values that you must specify for an operand. Type the values in the line of code in the order indicated. Type parentheses wherever they appear in a line of code. |
| *hh:mm:ss* | Indicates hours, minutes, seconds |
| RET | Indicates that you should press the Return key. |
| Ctrl/Z | Indicates that you should press the `z` key while holding the `Ctrl` key. |

## Acronyms

The following acronyms appear throughout this manual.

| | |
|---|---|
| API | Digital SNA Application Programming Interface for OpenVMS software |
| LU | Logical unit |

| | |
|---|---|
| LU0 | Logical unit type 0 |
| PLU | Primary logical unit |
| PU | Physical unit |
| RH | Request/response header |
| RU | Request/response unit |
| SLU | Secondary logical unit |
| SNA | IBM's Systems Network Architecture |
| SSCP | System services control point |

## Terminology

When this manual refers to the OpenVMS application, it means the application the user writes.

# 1 Introduction

The Digital SNA Application Programming Interface (API) for OpenVMS allows you to develop OpenVMS applications that exchange messages with cooperating applications on an IBM host. To exchange these messages, the OpenVMS application requires support from the API to establish a session with IBM logical units (LU), that conforms to function management (FM) and transmission services (TS) profiles 3 and 4. The API communicates with either the DECnet SNA Gateway-ST, the DECnet SNA Gateway-CT, the Digital SNA Domain Gateway-CT, the Digital SNA Domain Gateway-ST, the Digital SNA Peer Server or OpenVMS SNA software to provide support for the LU session.

**Note**

Unless stated otherwise, the term SNA Gateway refers to the DECnet SNA Gateway-ST, the DECnet SNA Gateway-CT, the Digital SNA Domain Gateway, the Digital SNA Peer Server, or the OpenVMS SNA (OpenVMS VAX Version 6.1 and Version 6.2 only) when used in this manual.

---

The API is one of three Digital SNA OpenVMS programming interface products. The remaining two products include the following:

- Digital SNA 3270 Data Stream Programming Interface for OpenVMS
  The Digital SNA 3270 Data Stream Programming Interface provides support to OpenVMS applications that need to establish a session with an LU type 2.
- Digital SNA APPC/LU6.2 Programming Interface for OpenVMS
  The Digital SNA APPC/LU6.2 Programming Interface provides support to OpenVMS applications that need to establish a session with an LU type 6.2.

Both of these products also enable you to exchange messages with cooperating applications on an IBM host.

To use the API, you must understand SNA well enough to write the code for performing the Transmission Control layer and the layers of SNA above that. In addition, your OpenVMS application must be able to operate as the secondary logical unit (SLU) while the cooperating IBM application operates as the primary logical unit (PLU).

## 1.1 API Features

Using the API, an OpenVMS application can perform the following functions:

- Communicate with logical units 0, 1, 2, and 3, using FM and TS profiles 3 and 4.
- Use different character sets
- Access various IBM equipment
- Restart a session after a failure at exactly the point where the session failed. (For more information about restarting a session after a failure, see *CICS/OS/VS IBM 3790/3730/8100 Guide*, Order No. SC33-0159 and *IMS/VS Version 1 Programming Guide for Remote SNA Systems*, Order No. SH20-9054.)
- Use customized applications

The API provides many services to help you write your application. The API

- Supports LU types 0, 1, 2, and 3, using FM and TS profiles 3 and 4.
- Provides functions common to all LU types.
- Does not include functions limited to a specific LU.
- Provides access to SNA common network functions performed by the SNA Gateway, such as Path Control and LU Services Manager functions.
- Provides protocol verification software to ensure that neither the IBM nor the OpenVMS side of the application violates the agreed set of protocols.
- Establishes sessions.
- Creates request/response units (RUs) for transmitted commands and data.
- Constructs request/response headers for RUs.
- Groups RUs into chains.
- Changes requests to responses.
- Generates sequence numbers.

In general, you are responsible for providing the software logic for sending, receiving, and interpreting the SNA protocol.

## 1.2 IBM and SNA Concepts

To use the API, you must understand the following:

- IBM's SNA protocols.
- The subsystem (CICS, for example) running on the IBM host.
- The protocol that will operate in the SNA Function Management (FM) layer. For instance, to write an application that requires the support of LU type 1, you need to know about the LU1 protocols, such as the SNA character set.

## 1.3 Common Interface Applications

You can use the API to write applications that make up the SLU half-session partner in an LU-LU session. For example, you can

- Perform file-transfer functions.
- Communicate with LU types 0, 1, 2, and 3, using FM and TS profiles 3 and 4.
- Emulate a terminal.
- Perform remote job entry (RJE).

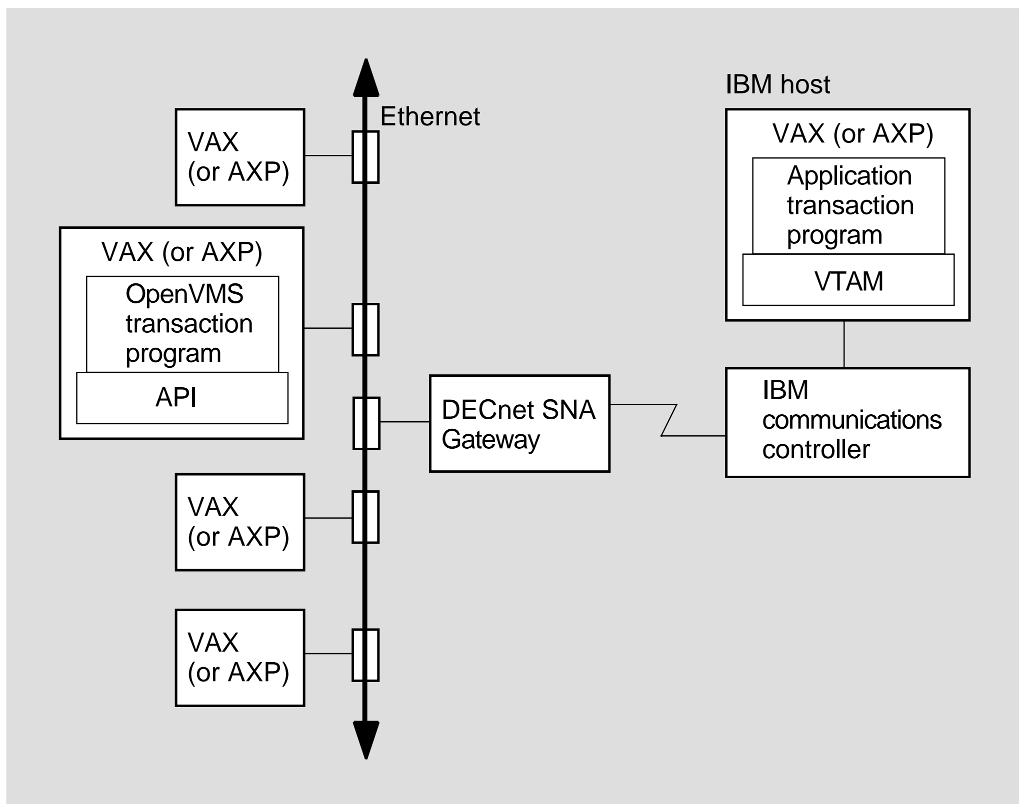
You can write applications to do a variety of tasks, but your application is responsible for checking protocols and status. The API allows you to tailor your applications to your needs.

For example, National Widget Sales, Inc., has IBM 8100 Information Systems distributed throughout the company. Because of the limited storage on the 8100, National Widget stores its files on an IBM host at company headquarters. When a branch office or a department needs a file, it transfers the file from the host, processes the file locally, and returns it to the host.

The 8100 uses the Distributed System Executive (DSX) to implement LU type 1 protocols to transfer files. Instead of using the 8100, a branch office or department could run an LU0 API application on OpenVMS VAX to access the files stored on the IBM host. By using the API, the OpenVMS VAX user can implement the LU1 protocols necessary for transferring files to or from the

IBM host. Figure 1–1 illustrates the Digital SNA Network environment. Note that this DECnet environment could also be TCP/IP.

**Figure 1–1 Digital SNA Network**



# 2 Concepts and Terms

The Digital SNA Application Programming Interface (API) for OpenVMS consists of procedures that a user-written OpenVMS program can call to request the following operations for an LU-LU type 0 session:

- Initiate a request to establish a session with an IBM application
- Respond to a session request initiated by an IBM application
- Transmit an SNA request unit containing user data, a data flow control request, or a session control request to an IBM application
- Receive an SNA request unit containing user data, a data flow control request, or a session control request from an IBM application
- Transmit and receive SNA response units
- Terminate a session
- Reestablish a session that has been temporarily terminated by the PLU with an UNBIND type 2 (hold resources) request

After a short description of the characteristics of an LU0 session, this chapter describes the above-mentioned procedures in more detail.

## 2.1 What is an LU-LU Type 0 Session?

As defined by IBM SNA terminology, an LU-LU session is the logical connection between two LUs. An LU-LU type 0 session uses the following subsets, or profiles, of communication functions:

- Any function management (FM) profile for an LU-LU session. An FM profile defines a commonly used subset of SNA-defined data flow control functions and certain other functions.
- Any transmission subsystem (TS) profile for an LU-LU session. A transmission subsystem profile defines a subset of SNA-defined transmission control functions and certain other functions.

- A set of end-user or product-defined protocols to augment or replace SNA presentation services and other high-level functions.

## 2.2 Establishing an LU-LU Session

Before end-users of an SNA network can exchange messages, their respective LUs must first establish an LU-LU session according to SNA protocol.

In general, any LU can issue a request to the system services control point (SSCP) for a session with another LU. To do this, the requesting LU sends the SSCP an initiate self (INIT-SELF) request that specifies the LU with which it wants to have a session. The SSCP selects one of the LUs as the primary LU (PLU) for the session and the other as the secondary LU (SLU). The SSCP then sends the PLU a control initiate (CINIT) request. The PLU, in turn, sends a BIND request to the SLU, proposing the conditions of the session. This request can be negotiable or nonnegotiable. If it is nonnegotiable, the SLU examines the BIND request and simply accepts or rejects the session. If it is negotiable, the SLU can accept or reject a session outright, or accept a session with specified changes in the proposed conditions.

In LU type 0, or LU0, sessions involving the SNA Gateway the OpenVMS application is always the SLU. This means that the OpenVMS application is always the receiver, never the sender, of the BIND request.

The OpenVMS application can issue two kinds of requests to establish an LU0 session: an active connect request and a passive connect request.

### 2.2.1 Issuing an Active Connect Request

An active connect request informs the API that the OpenVMS application wants to send an INIT-SELF request to the SSCP to initiate a session with a specified IBM application.

To issue an active connect request, the OpenVMS application calls the SNALU0$REQUEST_CONNECT procedure.

Input parameters include the following information that the application passes to the API:

- **An active/passive connection indicator**

  A value indicating that this is an active session request.

- **A SNA Gateway node or host name**

The name of the Gateway DECnet node or TCP/IP host name through which the OpenVMS application wishes to establish the session. This is an optional parameter. For OpenVMS SNA, set this parameter to equal an ASCII 0. If it is omitted, the API assumes you are requesting a connection via OpenVMS SNA.

- **IBM access information**

  A logical name associated with a list of default information required to gain access to the IBM host. This is an optional parameter. If it is omitted, the parameter list must explicitly provide the required values. For details on access names and IBM access information, see Section 3.3.

- **Address of an asynchronous event procedure**

  The address of a user-written procedure that the API can use to inform the OpenVMS application that an asynchronous event has occurred. For details on asynchronous events and the user-written notification procedure, see Chapter 3.

- **Address of a completion procedure**

  The address of a user-written procedure that the API can use to inform the OpenVMS application that the SNALU0$REQUEST_CONNECT procedure has completed.

Output parameters for the SNALU0$REQUEST_CONNECT procedure include locations to receive the following information from the IBM system:

- **A session identifier**

  A location to receive a unique identifier assigned by the API to the session. Each time the application issues a request to send or receive a message on this session, terminate the session, reconnect the session, or obtain information about the session, the parameter list for the call must include the session identifier. This is a required parameter.

- **BIND buffer**

  A buffer to receive the BIND request sent by the IBM application (the PLU) to establish the session. This is an optional parameter. If the application does not provide a buffer, the API unconditionally accepts the session.

- **I/O status block**

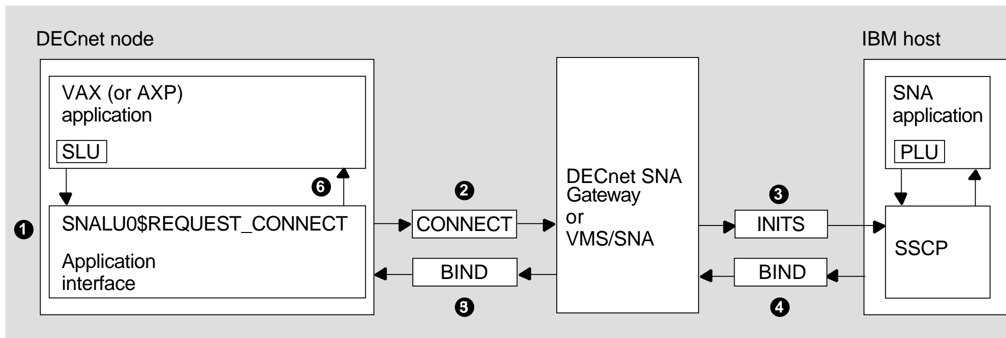
  A quadword status block to receive status information from the API. This is an optional parameter.

For a complete list of parameters for the SNALU0$REQUEST_CONNECT procedure, see Section 5.3.

Figure 2–1 illustrates the following steps taken during a typical active connect request. Note that this DECnet environment could also be TCP/IP.

1. The OpenVMS application calls the SNALU0$REQUEST_CONNECT procedure, setting the active request indicator and providing the other required parameters.
2. The API sends a connect request to the SNA Gateway.
3. The SNA Gateway sends an INIT-SELF request to the SSCP. The SSCP notifies the PLU.
4. The PLU sends a BIND request to the SNA Gateway.
5. The SNA Gateway sends the BIND request to the API.
6. The API passes the BIND to the OpenVMS application, along with an identification (ID) value for the session.

**Figure 2–1 An Active Connect Request**




## 2.2.2 Issuing a Passive Connect Request

A passive connect request informs the API that the OpenVMS application is ready to engage in a session initiated by an IBM application.

To issue a passive connect request, the OpenVMS application calls the SNALU0$REQUEST_CONNECT procedure and sets the active/passive connection indicator to request a passive connect. The OpenVMS application must also specify the SLU number (via the "access name" or "session address" parameter) to tell the API which session to listen for. The remaining parameters are the same as for an active request.

A typical passive connect request for a session includes the following steps:

1. The OpenVMS application calls the SNALU0$REQUEST_CONNECT procedure, setting the passive request indicator and providing the other required parameters.
2. The API sends a message to the SNA Gateway indicating that the OpenVMS application is ready to receive a session request initiated by the IBM application.
3. At some point, the SNA application sends a BIND request to the SNA Gateway.
4. The SNA Gateway sends the BIND request to the API.
5. The API passes the BIND request to the OpenVMS application, along with a value to identify the session.

| Input Parameters | Meaning |
|---|---|
| A passive connection indicator | A boolean flag indicating that the OpenVMS application is ready to receive a BIND request from an IBM application. This is a required parameter. |
| A Gateway DECnet node or TCP /IP host name | The name of the Gateway DECnet node or TCP/IP host name through which the OpenVMS application wishes to establish the session. This is an optional parameter. If it is omitted, the API obtains a default Gateway node. |
| IBM access information | A logical name associated with a list of default information (defined by the Gateway manager) required to gain access to the IBM host. This is an optional parameter. If it is omitted, the parameter list must explicitly provide the required values. For details on access names and IBM access information, see Section 3.3. |
| Address of an asynchronous event procedure | The address of a user-written procedure that the API can use to inform the OpenVMS application that an asynchronous event has occurred. This is an optional parameter. For details on asynchronous events and the user-written notification procedure, see Section 3.6. |

| Input Parameters | Meaning |
|---|---|
| Address of a completion procedure | Address of a user-written procedure that the API can use to inform the OpenVMS application that the SNALU0$CONNECT procedure has completed. |

| Output Parameters | Meaning |
|---|---|
| A session identifier | A location to receive a unique session identifier that the OpenVMS application will use in all subsequent references to the session. Each time the application issues a request to send or receive a message, terminate a session, reconnect a session, or obtain information about an active session, the parameter list for the call must include a session identifier. This is a required parameter. |
| BIND buffer | A buffer to receive the BIND request sent by the PLU to establish the session. This is an optional parameter. |
| I/O status block | A quadword status block to receive status information from the API. This is an optional parameter. |

### 2.2.3 Accepting or Rejecting a BIND Request

If the OpenVMS application provides a buffer, the API places in it the negotiable or nonnegotiable BIND request received from the IBM application. The OpenVMS application examines the BIND request and either accepts or rejects the session outright or, if negotiable, accepts it with specified changes. The BIND request states whether the request is negotiable. The OpenVMS application accepts or rejects a BIND request in one of the following ways:

- To accept a session proposed in a negotiable or nonnegotiable BIND request, the OpenVMS application calls the SNALU0$TRANSMIT_RESPONSE procedure and specifies a positive response as described in Section 2.8.
- To reject a session proposed in a negotiable or nonnegotiable BIND request, the OpenVMS application calls the TRANSMIT_RESPONSE procedure and specifies a negative response as described in Section 2.8.
- To accept a session proposed in a negotiable BIND request on the condition that certain parameters are changed, the OpenVMS application calls the TRANSMIT_RESPONSE procedure, specifies a positive response, and modifies the BIND buffer to reflect these changes.

If the OpenVMS application does not specify a buffer to receive the BIND request, the API accepts the session.

## 2.3 Using Request/Response Units

SNA defines two categories of messages, requests, and responses, known collectively as request/response units (RUs). The acronym RU refers to either a request unit or a response unit.

A request can consist of data that one application wants to send to another, or it can consist of control information. Ordinarily, a response is simply a positive or negative acknowledgment of a request.

The procedure that the OpenVMS application uses to send request units is described in Section 2.5.

The procedure for sending response units is described in Section 2.7.

The procedure for receiving request units and response units is described in Section 2.8.

## 2.4 Sending Request Units

In an SNA network, LUs exchange data and control messages in packages called request units. As defined by SNA, a request unit is a message of a specific size consisting of user data or control information. The maximum size that can be sent and received is specified in the BIND request used to establish the session and is enforced by the API.

As part of the control information required for transmission, each request unit carries a request header and a sequence number or other unique identifier.

### 2.4.1 The Request Header

In transmissions involving the SNA Gateway, the API is responsible for constructing a request header (RH) for each request unit and setting the indicators as required. The OpenVMS application can control the way the API sets certain RH indicators. These include:

- Chaining indicators
- Change direction indicator
- Bracketing indicators
- Response indicators

#### 2.4.1.1 Chaining Indicators

In an SNA network, a chain is a recoverable unit of transmission. All request units (with their attached headers) travel in chains. A chain may consist of multiple request units or a single request unit. The following list provides information about chaining indicators that delimit the beginning and end of different kinds of chains:

- In a chain consisting of multiple request units, the first request carries a begin chain indicator (BCI) in the request header to indicate that it is the first request in the series. The last request unit in the chain carries an end chain indicator (ECI) to indicate that it is the last request in the series.
- In a chain consisting of a single request unit, the request unit carries both the BCI and the ECI.

In transmissions involving the SNA Gateway, the API handles chaining by default for the OpenVMS application. If desired, however, the OpenVMS application can control the setting of the chaining indicators. The OpenVMS application controls the setting of the ECI by means of the *more-data* parameter in the TRANSMIT_MESSAGE procedure.

#### 2.4.1.2 Change Direction Indicator

If the session is using half duplex flip-flop mode for sending and receiving chains (see Section 4.2.2.1), the last request unit in a transmitted chain normally carries a change direction indicator (CDI) in the RH. The CDI indicates that the OpenVMS application has finished sending a chain and that it is now the IBM application's turn to send a chain. When brackets are used, the CDI has no meaning between brackets.

On each chain that the OpenVMS application submits for transmission in a half duplex flip-flop session, the API sets the CDI by default. If the OpenVMS application wishes to send consecutive multiple chains, it must tell the API not to set the CDI. To do this, the OpenVMS application uses the "turn retain" parameter described in Section 2.6.

#### 2.4.1.3 Bracketing Indicators

Chains of request units and their responses (see Section 4.2.6) can be grouped together within delimiters called brackets. Brackets define a group of chains as a complete unit of work. The following list provides information about bracket indicators that delimit the beginning and end of a bracket:

- The first request unit in the first chain carries a begin bracket indicator (BBI) in its RH.
- The first request unit in the last chain carries an end bracket indicator (EBI).

The BIND request specifies whether brackets will be used in the session and how the EBI is set.

In sessions involving the SNA Gateway, the API sets the BBI to indicate the beginning of a bracketed chain. If the BIND request allows the SLU to send the EBI, the OpenVMS application is responsible for setting the EBI.

**Note**

If bracketing is used, one LU is designated "first speaker" and can begin a bracket without asking permission of the other LU. The other LU becomes the "bidder" and must ask permission by sending a BID request. If the SLU is specified as the "bidder," the OpenVMS application is responsible for sending the BID request and receiving a response. Certain IBM host applications allow the bidder to omit the BID and simply take a chance on setting the BBI in the first chain.

#### 2.4.1.4 Response Indicators

The OpenVMS application can specify the type of response required for a chain that it transmits. The response type can be none, definite, or exception. The response type can also include a number (1, 2, or 3) that conveys additional information to the IBM application. Normally, response type 1 is used and given no special meaning.

### 2.4.2 Sequence Numbers and Unique Identifiers

Each request unit that travels on the normal data flow in an SNA network carries a sequence number to identify its position in its chain. Each request unit that travels on the expedited flow carries a unique identifier.

In transmissions involving the SNA Gateway, the API is responsible for assigning a sequence number to each request unit that it sends to the IBM application on the normal flow. The API returns to the OpenVMS application the sequence number that it assigns to both the first request unit and the last request unit in the chain.

The API assigns a unique identifier to each request unit that it transmits on the expedited flow. The API returns this value to the OpenVMS application.

## 2.5 Transmitting User Data in Request Unit Chains

To transmit user data to an IBM application, the OpenVMS application calls the SNALU0$TRANSMIT_MESSAGE procedure and provides a buffer containing the data.

By default, the API packages the user data into one or more request units of the maximum size possible, adds RHs and sequence numbers, and transmits the units to the IBM host over the normal flow as a complete chain. The API sets the BCI in the RH of the first request unit and the ECI in the RH of the last request unit.

Input parameters for the SNALU0$TRANSMIT_MESSAGE procedure can pass the following information to the API:

- **Session identifier**

  A value specifying the session on which the API is to transmit the request unit chain. (This is the value returned by the SNALU0$REQUEST_CONNECT procedure when the session is established.)

- **Transmit buffer**

  The buffer containing the user data that the OpenVMS application wants to transmit to the IBM host.

- **Message class indicator**

  A value indicating whether the buffer contains formatted or unformatted user data.

- **More data indicator**

  A TRUE/FALSE flag indicating whether the application intends to add more request units to the current chain (by issuing one or more additional calls to the SNALU0$TRANSMIT_MESSAGE procedure).

  1. If FALSE (no more data), the API sets the ECI in the last request unit it constructs from the contents of the buffer. The chain is complete. The next time the OpenVMS application calls the TRANSMIT_MESSAGE procedure, the API begins a new chain, setting the BCI in the first request unit.
  2. If TRUE (more data), the API does not set the ECI and the chain remains open. The next time the OpenVMS application calls the TRANSMIT_MESSAGE procedure, the API adds requests units to the existing chain.

- **End bracket indicator**

A TRUE/FALSE flag telling the API whether to set the EBI on the request unit carrying the BCI for the last chain in the bracket. This parameter is valid only if the BIND request for the session specified bracketing and the SLU is allowed to send EBI.

1. If TRUE, the API sets the EBI on the request unit carrying the BCI. The bracket is complete.
2. If FALSE, the API does not set the EBI and the bracket remains open.

- **Response type indicator**

  A value indicating the type of response to be supplied by the IBM application to the request unit carrying the ECI.

  To specify "no response," you must set the *resp-type* parameter for each TRANSMIT call that pertains to the chain to SNALU0$K_RSP_NONE. The API sets the response indicator for each RU in the chain to indicate "no response." The default is "no response."

  To specify "exception response," you must set the *resp-type* parameter for each TRANSMIT call that pertains to the current chain to SNALU0$K_RSP_RQE*n* (where *n*=1, 2, or 3). The API sets the response indicator for each request unit in the chain to indicate "exception response."

  To specify "definite response," you must set the *resp-type* parameter for each TRANSMIT call that pertains to the current chain as follows:

  - If the transmit buffer contains either the beginning or the middle of a chain, *resp-type* equals SNALU0$K_RSP_RQE*n* (where *n*=1, 2, or 3). You must specify the same exception response type number for each TRANSMIT call that pertains to the current chain.
  - If the transmit buffer contains a complete chain or the end of a chain, *resp-type* equals SNALU0$K_RSP_RQD*n*, where *n* is the same type number you specified for the exception response.

  The API sets the first and middle RUs to indicate "exception response" and sets the last RU to indicate "definite response."

- **Turn retain indicator**

  A TRUE/FALSE flag telling the API whether to set the CDI on the transmitted request unit carrying the ECI. This parameter is valid only for sessions using the half duplex flip-flop mode of message exchange.

  1. If FALSE, the API sets the CDI on the request unit carrying the ECI. The IBM application can now send a message.

2. If TRUE, the API does not set the CDI. The OpenVMS application can call the TRANSMIT_MESSAGE procedure to send more data.

Output parameters include the following locations to receive information from the IBM application:

- **First and last sequence number**

  Locations to receive the sequence number that the API assigned to both the first and the last request in the transmitted chain. (For a single-unit chain, the API returns the same value to both locations.)

In the simplest type of data transmission involving the SNA Gateway, the following steps occur:

1. The OpenVMS application calls the SNALU0$TRANSMIT_MESSAGE procedure and passes a buffer. The application indicates whether the buffer contains formatted or unformatted user data.
2. The API creates the first request unit from the user data, adds a header, and assigns a sequence number.
3. The API sets the BCI in the header for the first request unit and transmits the unit to the IBM host over the normal flow.
4. As long as data remains in the buffer, the API continues to create request units of the maximum size possible, add headers, assign sequence numbers, and send the units to the IBM host.
5. The API creates the last request unit from the remaining data in the buffer, adds a header, and assigns a sequence number.
6. The API sets the ECI in the header for the last request unit for transmission and sends the unit to the IBM host.

If the *more-data* parameter for the TRANSMIT_MESSAGE call equals TRUE, the API does not set the ECI on the last request unit. As a result, the chain remains open. When the application issues subsequent calls to the SNALU0$TRANSMIT_MESSAGE procedure to send user data, the API continues to add the data to the existing chain until the OpenVMS application issues a TRANSMIT_MESSAGE call with the *more-data* parameter set to FALSE.

## 2.6 Transmitting Control Information in Request Unit Chains

The OpenVMS application is responsible for exchanging data flow control and session control commands with the IBM application.

To transmit a data flow control or session control command to an IBM application, the OpenVMS application calls the SNALU0$TRANSMIT_MESSAGE procedure and supplies the command. The API then packages the command as a single request unit and transmits it to the IBM host over the appropriate flow as a single request unit chain.

Input parameters for the SNALU0$TRANSMIT_MESSAGE procedure include the following:

- **Session identifier**

  A value specifying the session on which the API is to transmit the request unit chain. (This is the value returned by the SNALU0$REQUEST_CONNECT procedure when the session is established.)

- **Transmit buffer**

  A data control command or session control command that the OpenVMS application wants to transmit to the IBM host.

- **Message class indicator**

  A value indicating whether the buffer contains a data flow control command or a session control command.

- **End bracket indicator**

  A TRUE/FALSE flag telling the API whether to set the EBI on this request unit. For session control requests, the EBI equals 0. For data flow control requests, the EBI is normally 0.

- **More data indicator**

  A TRUE/FALSE flag telling the API whether the application intends to add more request units to the current chain (by issuing one or more additional calls to the SNALU0$TRANSMIT_MESSAGE procedure).

- **Response type indicator**

  A value indicating the type of response to be supplied by the IBM application to the request unit carrying the ECI. This flag is required for a single-request chain. The default is no response requested.

- **Turn retain indicator**

A TRUE/FALSE flag telling the API whether to set the CDI on the request unit. For session control requests, the CDI equals 0. For data flow control requests, the CDI is normally 0.

The complete parameter list specifies locations to receive the following information from the API:

- **Unique identifier**

  A location to receive a unique identifier assigned by the API to the single request unit in the chain. This is the same as the location to receive the first sequence number. The API also returns this identifier to the location specified for the last sequence number.

In the simplest type of command transmission involving the SNA Gateway, the following events occur:

1. The OpenVMS application calls the SNALU0$TRANSMIT_MESSAGE procedure and passes the command. The application indicates whether the command is a data flow control command or a session control command.
2. The API packages the command as a request unit, adds a header, and assigns a unique identifier.
3. The API sets the BCI and the ECI in the header and sends the unit to the IBM host as a single unit chain.

## 2.7 Transmitting Response Units

Unless the IBM application specifies that no response is required, the OpenVMS application must return a response to each request unit it receives. This includes a request that specifies an exception response. The API needs to know that it does not need to send a negative response and can therefore update its internal state tables.

If the application does not wish to generate a negative response, a response should only be issued to the last request unit in a chain. For example, this situation occurs when the SNALU0$RECEIVE_MESSAGE procedure returns a value of FALSE for the *more-data* parameter.

To send a response, the OpenVMS application calls the SNALU0$TRANSMIT_RESPONSE procedure.

Input parameters for the call include the following:

- **Session identifier**

A value specifying the session on which the API is to transmit the response. (This is the value returned by the SNALU0$REQUEST_CONNECT procedure when the session is established.)

- **Request buffer address**

  A buffer containing the received request to which the application is responding.

- **Request buffer size**

  A value indicating the size of the request buffer.

- **Response type**

  A value indicating whether the response is positive or negative.

- **Sense data**

  Sense data (IBM error codes) to be returned to the PLU if the response is negative.

## 2.8 Receiving Request/Response Units

During the course of a session, the OpenVMS application typically receives three types of messages from the IBM application:

- A single request unit or multiple request units containing user data and comprising either a complete or a partial chain.
- A single request unit containing a data flow control command or a session control command and comprising either a complete or a partial chain.
- A single response unit.

To receive any of these types of messages, the OpenVMS application calls the SNALU0$RECEIVE_MESSAGE procedure and provides a buffer.

The OpenVMS application passes the location of a data buffer to the API by descriptor. Most languages handle descriptors transparently. (Exceptions are BLISS, MACRO, and C.) The application simply passes the name of the buffer to the API procedure.

For applications that specify descriptor types, class S (static) descriptors and class D (dynamic) descriptors, as described in the "VAX Procedure Calling and Condition Handling Standard," are recommended for passing the location of a data buffer to the API.

If the parameter list specifies a class S descriptor, the API copies the message from its own buffer into the buffer pointed to by the descriptor.

If the parameter list specifies a class D descriptor, the API fills in the descriptor with a pointer to its own buffer, which contains the received message. The API also places the size of the buffer in the descriptor. It returns the actual length of the data as a separate parameter. Note that the class D descriptor provides a more efficient means for an application to receive a normal flow message than does the class S descriptor, which requires the API to copy the received message from one location to another.

The OpenVMS application must not write into a class D descriptor that it uses to pass a receive buffer. Class D descriptors must be manipulated only by means of system library string routines.

If a class D descriptor is used, the OpenVMS application is responsible for ensuring that buffer space is returned to free memory once it is no longer needed. To return a buffer to free memory, the application can call the system library procedure LIB$SFREE1_DD or STR$FREE1_DX. If the application fails to free the buffer space, the process will quickly run out of virtual memory, producing unpredictable results. Typically, calls to API procedures will return with an insufficient virtual memory error.

Required and optional input parameters include the following information that the application supplies to the API:

- **Session identifier**

  A value specifying the session on which the API is to receive the request unit chain or response unit. (This is the value returned by the SNALU0$REQUEST_CONNECT procedure when the session is established.)

- **Receive buffer**

  A buffer to contain the received message. If the buffer is not large enough to contain all the RUs in the chain, the API passes as many units as it can.

  **NOTE**

  This buffer must be large enough to contain at least one RU plus a 7-byte header.

Output parameters for the SNALU0$RECEIVE_MESSAGE procedure include the following locations to receive information from the API:

- **Message length**

  A location to receive from the API a value indicating the actual length of the message including the 7-byte header.

- **Request/response indicator**

  A location to receive a value from the API indicating whether the buffer contains a request unit chain or a response unit.

- **Message class indicator**

  A location to receive a value from the API indicating the class of the received command. Command classes are described in Section 3.4.

- **Message type indicator**

  If the receive buffer contains a data flow control or session control command, the API returns a value to this location indicating the type of the received command. Command types are described in Section 3.4.

- **More data indicator**

  A location to receive a TRUE/FALSE flag indicating whether the last request unit in the receive buffer carries the end chain indicator.

- **Flow indicator**

  A location to receive a value indicating whether the received chain arrived on the normal flow or the expedited flow.

- **Alternate code indicator**

  A location to receive a TRUE/FALSE flag indicating the presence of a character set other than EBCDIC in the received chain.

- **Bracket indicators**

  Two locations to receive TRUE/FALSE flags indicating that the first request carries the BBI and that the last request carries the EBI. The BBI and EBI equal 0.

- **Response type indicator**

  A location to receive a value indicating the type of response requested by the received chain.

- **End data indicator**

  A location to receive a TRUE/FALSE flag indicating whether the received chain carries the CDI. The CDI equals 0.

- **First and last sequence numbers**

  Locations to receive either the sequence numbers for the first and last request in a received chain or the unique identifier for a single-unit chain containing a control message.

## 2.9 Terminating a Session

To terminate an LU-LU session, the OpenVMS application calls the SNALU0$REQUEST_DISCONNECT procedure and specifies the session.

As a result of this call, the SNA Gateway sends an unconditional TERMINATE SELF request to the SSCP, and the API deallocates all resources allocated to the session. If the session is already inactive when the application calls the SNALU0$REQUEST_DISCONNECT procedure, the API deallocates resources but the SNA Gateway does not send a TERMINATE SELF message.

## 2.10 Reestablishing a Session

In an SNA network, the PLU can terminate a session by sending the SLU an UNBIND type 2 (hold resources) request to indicate that another BIND request will soon follow to reestablish the session. The application receives an UNBIND type 2 via asynchronous event notification. An UNBIND type 2 requests the SLU to reserve the resources allocated to the session.

To respond to an UNBIND type 2 request sent from the IBM application, the OpenVMS application calls the SNALU0$REQUEST_RECONNECT procedure. The API places the session in a BIND_PENDING state and, if possible, reserves the resources allocated to the session.

# 3 API Features

The Digital SNA Application Programming Interface (API) for OpenVMS provides features to assist you in writing and executing your application. These features include the following:

- Status information
- Synchronous and asynchronous operation
- IBM access names
- Message classes and types
- State machine information
- Asynchronous event notification

## 3.1 Status Codes

The API uses two mechanisms to return status codes to the OpenVMS application:

- Function value returns
- An I/O status vector

For a description of all the status codes returned by the API, see Appendix F.

### 3.1.1 Function Value Returns

When an API 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 API procedure, you must check this status value. The value in the low-order word indicates either that the procedure completed successfully or that some specific error prevented the procedure from performing all or some of its functions.

Each high-level language provides some mechanism for testing the return status value in R0. Often you need to check only the low-order bit, such as by a test for TRUE (success or informational 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 procedures return status messages via a data structure called a status vector. Status vectors provide complete information about error conditions and use a format identical to the message vector format used by OpenVMS. Status vectors supply:

- Success messages
- Warning messages
- Error messages
- Informational messages
- Severe error messages

Figure 3–1 shows the format of the status vector. The status vector can contain one or more error messages, depending upon the kind of error that occurred.

If an error occurs, each component of the network involved can pass a message to the API. The API uses this information to build the status vector. The completed status vector is available to the application program at call completion.

Usually, the application displays the error via 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. 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. 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 |

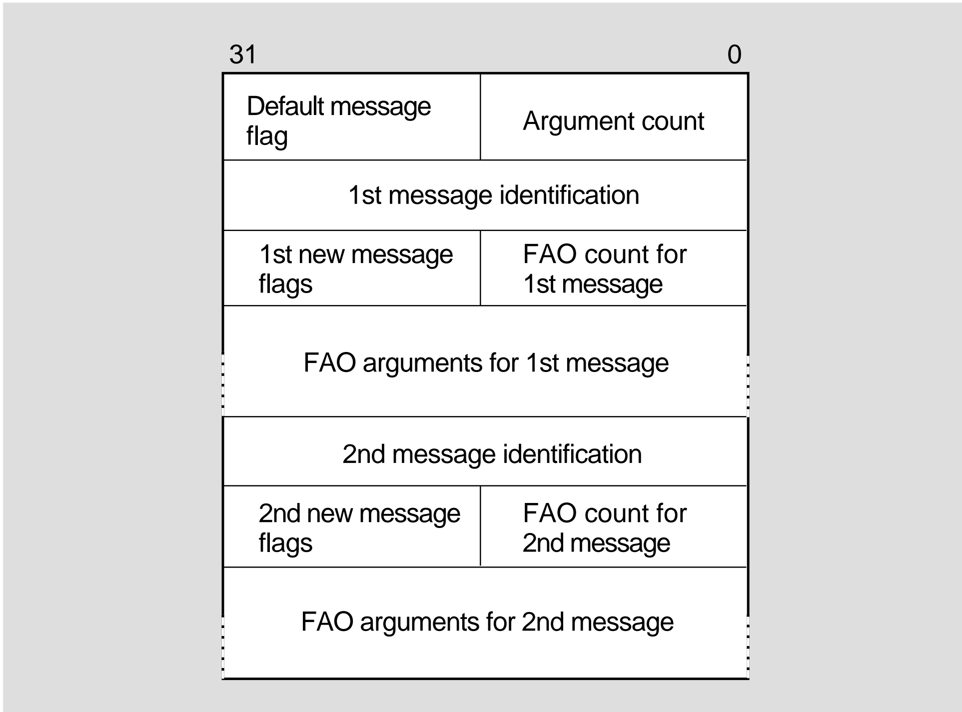
The application does not have to signal, however. The programmer can choose other options, such as having the program attempt to recover from the error.

For further information about dealing with errors, see under "Condition Handling" and "$PUTMSG" in the *OpenVMS System Services Reference Manual*. See under "LIB$CALLG," "LIB$SIGNAL," and "LIB$STOP" in the *OpenVMS Run-Time Library Routines Reference Manual*).

**Note**

You must define a vector of minimum size, using the SNALU0$K_MIN_STATUS_VECTOR literal, and provide a descriptor pointing to it in each procedure call. The API can then fill in the vector at the completion of the operation.

**Figure 3–1 Status Vector**




The following list provides a description of the fields in the status vector.

- Argument count

  Specifies the total number of longwords in the status vector.

- 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 this call until a new set of flags (if any) is specified. That is, each "new message flags" field specified sets a new default.

  Bits 20 through 31 must be zeros.

- Message identification

32-bit numeric value that uniquely identifies this message. Messages can be identified by symbolic names defined for system return status codes, VAX–11 RMS status codes, and so on.

- FAO count

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

- New message flags

  New mask for the $GETMSG flags, defining a new default for this message and all subsequent messages.

- FAO arguments

  FAO arguments required by the message.

## 3.2 Synchronous and Asynchronous Operation

An application that calls an API procedure can specify two modes of operation: synchronous and asynchronous.

### 3.2.1 Synchronous Mode

In synchronous, or wait, mode, the following steps occur:

1. The OpenVMS application 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 API returns status information immediately as a function value and with further information in the status vector.
3. The API sends the request to the SNA Gateway and suspends the OpenVMS application.
4. The SNA Gateway performs the operation and sends the result to the API.
5. The API procedure returns a function value to indicate the success or failure of the operation. The procedure also places the status code and further information in the status vector. The application resumes execution.

A synchronous call has the following general format:

SNALU0$*procedure*_W (*parameters*)

where

| | |
|---|---|
| SNALU0$procedure_W | is the name of the procedure, and *parameters* is a list of information needed to perform the requested operation. |

### 3.2.2 Asynchronous Mode

In asynchronous mode, the application issues a call to request an operation and immediately resumes execution. It does not wait for the operation to be completed. For this reason, applications that call procedures asynchronously must either specify an event flag or provide a completion procedure that the API can call to indicate that the SNA Gateway has completed its attempt to perform the operation.

The API completion procedure is an asynchronous system trap (AST). For information about the AST, event flag services, and AST services, see the *OpenVMS System Services Reference Manual*).

**Note**

Use system service calls to enable and disable ASTs. The API is based upon AST completion. If you disable ASTs and leave them disabled, no requests will be able to complete.

---

An asynchronous call involves the following steps:

1. The application issues a call to an API procedure to request an operation.
2. The procedure immediately returns a status code as a function value. If the application issues the call successfully, step 3 occurs. If the call fails, step 4 occurs.
3. The procedure returns a function value indicating success of the call, and the application resumes execution. At the completion of the operation, the API will perform the following steps:
   - Fill in the status vector with completion information indicating success or failure
   - Set an event flag
   - Call a completion procedure if one was specified to inform the application that the API has finished its attempt to perform the requested operation

4. The procedure returns a function value indicating that the call was unsuccessful. The procedure also places the status code and other information in the status vector. The API does not attempt to perform the operation. The application resumes execution.

**Note**

The notify routine is not interrupted by completion ASTs rather, the ASTs are queued and serviced sequentially. Similarly, the completion ASTs are not interrupted by the notify routine.

---

An asynchronous call has the following general format:

SNALU0$*procedure* (*parameters*)

where

| | |
|---|---|
| SNALU0$procedure | is the name of the procedure, and *parameters* is a list of information needed to perform the requested operation. The user-written procedure that an API procedure calls to indicate that the SNA Gateway has completed its attempt to perform a requested operation has the following calling format: *procedure* (*ast-par.rlu.r*) |
| where | |
| procedure | is the name of the user's routine that is being called. (*procedure* is specified as the *ast-addr* parameter in an asynchronous call to an API procedure.) |
| ast-par | is a parameter passed to the user-written procedure. You can use the *ast-par* to provide a pointer to the *session-id* or a data structure containing the *session-id* in multisession applications. (*ast-par* is specified as the *ast-par* parameter in an asynchronous call to an API procedure.) |

**Note**

In both synchronous and asynchronous calls, the application is responsible for providing an event flag number in the parameter list for use by the API procedure. If the application omits the event flag number, the API assumes event flag 0. You should always specify a nonzero event flag because zero is often the default and may result in programming errors.

---

## 3.3 Supplying Access Information to the IBM Host

In order to establish a session with an IBM application, the OpenVMS application must supply the following information to the IBM host:

- **PU identification** - A value identifying the Gateway Physical Unit (PU) (for example, LC-0) or the OpenVMS/SNA PU (SNA-0) used to establish the session. This information is supplied only to Gateway-ST and Gateway-CT style gateways.
- **Application name** - An ASCII character string identifying the PLU application (for example, CICS) that you want to connect to in the IBM host.
- **Session address** - A value indicating the SLU address that you want to use to establish a session with the IBM host. The value you specify should be the same as the session address (LOCADDR parameter) defined in the NCP (and VTAM) definitions which the SNA Gateway you are using is connected to. This information is not used when specifying LU identification information.
- **Logon mode name** - An ASCII character string specifying an entry in a logon mode table that gives a set of BIND parameters for the session. (See your VTAM system programmer for more information.)
- **IBM user identification** - A value identifying the user to the IBM session. This value is inserted into the "requester ID" field of the INIT-SELF message.
- **IBM password** - A string associated with the IBM user ID. This value is inserted into the "password" field of the INIT-SELF message. (Some IBM applications require a password others do not.)
- **Optional user data** - Data passed to the IBM application. (The meaning of the data is IBM application dependent.)
- **LU identification** - A value identifiying the Gateway LU (for example, H010A00E) used to establish the session.

**Note**

VTAM normally ignores both the "requester ID" and "password" fields. See the IBM *Systems Network Architecture Format and Protocol Reference Manual: Architectural Logic*, Order No. SC30-3112, for further information.

The application supplies this information as parameters each time it issues a call to the SNALU0$REQUEST_CONNECT procedure.

The Gateway manager can define a complete or partial list of IBM access information and associate the list with an access name. If the application specifies the access name in the parameter list of a call to SNALU0$REQUEST_CONNECT, all IBM access information defaults to the values in the associated list. To override a value associated with an access name, specify a new value in the parameter list. For further information about IBM access information and access names, see the *DECnet SNA Gateway-ST Installation*, the *DECnet SNA Gateway-CT Installation*, or the *OpenVMS SNA Installation*.

## 3.4 Message Classes and Types

All SNA request units exchanged between the OpenVMS application and the IBM application are identified by class. There are four message classes: formatted user data, unformatted user data, data flow control, and session control.

Request units that contain a data flow control or session control command are also identified by type. The type indicates the control function that is being requested.

Class names have the following general format:

SNALU0$K_MCLASS_*class*

where *class* is one of the values listed in Table 3–1.

Type names have the following general format:

SNALU0$K_MTYPE_*type*

where *type* is one of the values listed in Table 3–1.

In Table 3–1 the facility code prefix (SNALU0$K_) has been left off the message types. For instance, cancel is actually SNALU0$K_MTYPE_CANCEL

**Table 3–1 Request Unit Classes and Types**

| Class | Type | Description |
|---|---|---|
| MCLASS_FORMATTED_FM | | Formatted user data |

(continued on next page)

**Table 3–1 (Cont.) Request Unit Classes and Types**

| Class | Type | Description |
|---|---|---|
| MCLASS_UNFORMATTED_FM | | Unformatted user data |
| MCLASS_DFC | | Data flow control request |
| | MTYPE_BID | Bid |
| | MTYPE_BIS | Bracket initiation stopped |
| | MTYPE_CANCEL | Cancel |
| | MTYPE_CHASE | Chase |
| | MTYPE_LUSTAT | LU status |
| | MTYPE_QC | Quiesce complete |
| | MTYPE_QEC | Quiesce at end of chain |
| | MTYPE_RELQ | Release quiesce |
| | MTYPE_RSHUTD | Request shutdown |
| | MTYPE_RTR | Ready to receive |
| | MTYPE_SBI | Stop bracket initiation |
| | MTYPE_SHUTC | Shutdown complete |
| | MTYPE_SHUTD | Shutdown |
| | MTYPE_SIG | Signal |
| MCLASS_SESSION_CONTROL | | Session control request |
| | MTYPE_RQR | Request recovery |
| | MTYPE_STSN | Set and test sequence number |
| | MTYPE_CLEAR | Clear data traffic |
| | MTYPE_SDT | Start data traffic |

**Note**

The symbol SNALU0$K_MCLASS_NETWORK_CONTROL exists, but the OpenVMS application will never see network control messages because they are used in SSCP to LU sessions.

## 3.5 State Machine Information

When the API is in session with IBM, it can, at any time, be in one of many states. You can learn which state the API is in by calling the SNALU0$EXAMINE_STATE procedure. The procedure returns the symbolic codes listed in Table 3–2:

**Table 3–2 Symbols Returned by the SNALU0$EXAMINE_STATE Procedure**

| Symbol | Meaning |
|---|---|
| *Session States* | |
| SNALU0$K_ST_SES_RESET | Session reset |
| SNALU0$K_ST_SES_P_ACTIVE | Active session pending |
| SNALU0$K_ST_SES_ACTIVE | Session active |
| SNALU0$K_ST_SES_P_RESET | Session reset pending |
| *Bracket State Manager* | |
| SNALU0$K_ST_BSM_BETB | Between brackets |
| SNALU0$K_ST_BSM_INB | In brackets |
| SNALU0$K_ST_BSM_P_BB | Between brackets pending |
| SNALU0$K_ST_BSM_P_INB | In brackets pending |
| SNALU0$K_ST_BSM_P_TERM_S | Pending bracket termination send |
| SNALU0$K_ST_BSM_P_TERM_R | Pending bracket termination receive |
| *Chain States* | |
| SNALU0$K_ST_CHAIN_BETC | Between chain |
| SNALU0$K_ST_CHAIN_INC | In chain |
| SNALU0$K_ST_CHAIN_PURGE | Purge chain |
| *Turn States* | |

(continued on next page)

**Table 3–2 (Cont.) Symbols Returned by the SNALU0$EXAMINE_STATE Procedure**

| Symbol | Meaning |
|---|---|
| SNALU0$K_ST_TURN_CONT | Contention |
| SNALU0$K_ST_TURN_CONT_S | Contention send |
| SNALU0$K_ST_TURN_CONT_R | Contention receive |
| SNALU0$K_ST_TURN_SEND | Send |
| SNALU0$K_ST_TURN_RCV | Receive |
| SNALU0$K_ST_TURN_RCV_81B | Received 081B sense code |
| SNALU0$K_ST_TURN_ERPS | Error recovery send |
| SNALU0$K_ST_TURN_ERPR | Error recovery receive |
| *Quiesce State* | |
| SNALU0$K_ST_QEC_RESET | Quiesce reset |
| SNALU0$K_ST_QEC_PEND | Quiesce pending |
| SNALU0$K_ST_QEC_QUIESCED | Quiesced |

## 3.6 Asynchronous Event Notification

The API provides a means of informing the application that one or more asynchronous events have occurred. This notification can take place at any point during a session. The asynchronous events that can occur include the following:

- A network communication error has been detected.
- The IBM host, the SNA Gateway has deliberately unbound the session or terminated the connection.
- The IBM host has violated the SNA protocol.
- The IBM host has sent an UNBIND type 2–reconnection pending.
- The PLU has sent a CLEAR request to reset the session, and the application has responded with +RSP.

The OpenVMS application can include a user-written notification procedure that the API calls each time one of these asynchronous events occurs during a session.

When you call the SNALU0$REQUEST_CONNECT procedure, use the *notify-rtn* parameter to indicate the user-written procedure. The notify routine can examine the event code (listed below) it receives and take action, such as returning a message to the application about the nature of the asynchronous event.

The calling format for the user-written procedure is as follows:

```
notify-rtn.zem.r (event-code.rlu.r,notify-parm.rlu.r)
```

where

*notify-rtn* is the name of the procedure specified in the connect call.

*event-code* is a symbolic code indicating the nature of the event.

- SNAEVT$K_COMERR
  A network communication error
- SNAEVT$K_TERM
  A deliberate termination of the link by the IBM host, or the SNA Gateway, (see Section 2.9)
- SNAEVT$K_UNBHLD An UNBIND type 2 sent by IBM–reconnection pending (see Section 2.10)
- SNAEVT$K_PLURESET The PLU reset the session by sending CLEAR (see Section 4.3.2.2)

**Note**

The *event-code* symbols are defined in SNALU0DEF.

notify-parm is an optional user-specified parameter to be passed to the notification procedure. You can use the *notify-parm* to provide a pointer to the *session-id* or a data structure containing the *session-id* in multisession applications. Passed by reference.

If one of the asynchronous events described above occurs, the following steps take place:

1. The API fills out the notify vector supplied by the OpenVMS application in the REQUEST_CONNECT procedure.
2. The API notifies the OpenVMS application of the asynchronous event by calling the user-written notify procedure.
3. The OpenVMS application reads the event code returned to the user-written procedure.

4. The OpenVMS application reads the notify vector for detailed information about the asynchronous event.

**Note**

The notify routine is not interrupted by completion ASTs rather, the ASTs are queued and serviced sequentially. Similarly, the completion ASTs are not interrupted by the notify routine.

---

The format and function of the notify vector are the same as the status vector (see Section 3.1.2).

Usually, the application signals an event via the system service call $PUTMSG. $PUTMSG translates the notify vector into a human-readable message that can be sent to a terminal or file.

**Note**

You must define a vector of minimum size using the SNALU0$K_MIN_NOTIFY_VECTOR literal, and provide a descriptor pointing to it in the SNALU0$REQUEST_CONNECT procedure call. The API can then fill in the vector.

---

# 4 SNA Functions

The responsibility for providing SNA functions in an LU-LU type 0 (LU0) session is divided between the SNA Gateway, the API, and the OpenVMS application.

SNA functions are grouped in "layers" of related functions (see Figure 4–1. This chapter summarizes the functions performed at each of the following layers and indicates which are handled by the OpenVMS application, which are handled by the API, and which are handled by the SNA Gateway.

- The LU Services layer (Section 4.1)
- The Data Flow Control layer (Section 4.2)
- The Transmission Control layer (Section 4.3)

## 4.1 The LU Services Layer

SNA defines two categories of LU services: LU network services and LU presentation services.

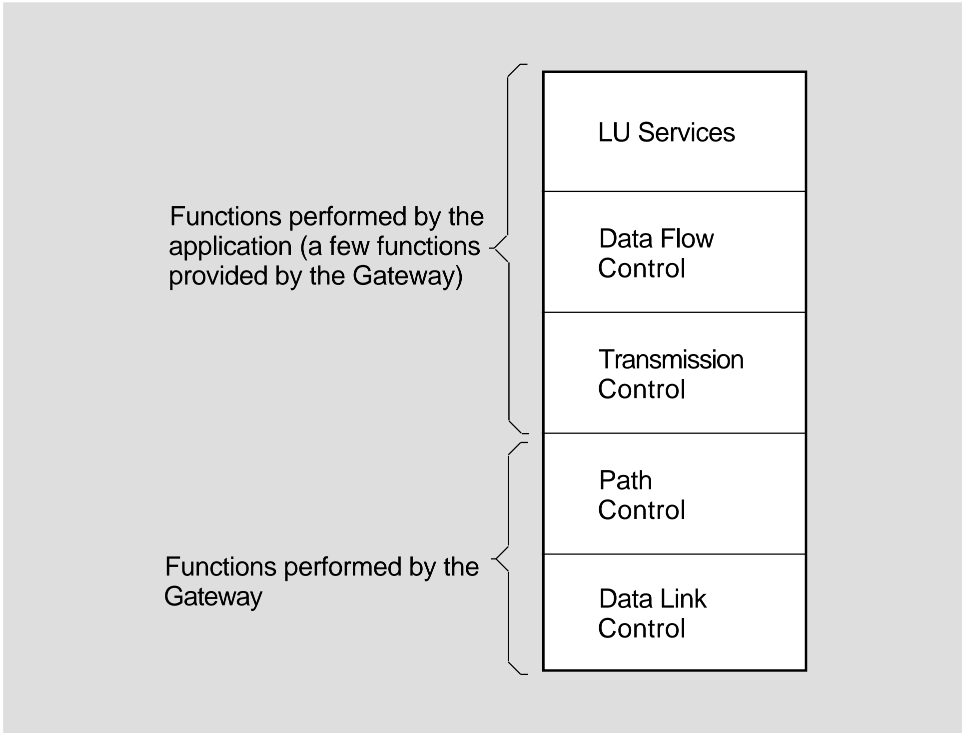
### 4.1.1 Network Services

LU network services include procedures for requesting initiation and termination of an LU-LU session.

To request initiation of a session, an LU sends an initiate self (INIT-SELF) request to the SSCP. In an LU0 session involving the SNA Gateway, the SNA Gateway sends INIT-SELF in response to an SNALU0$REQUEST_CONNECT call from the OpenVMS application.

To request termination, the LU sends a terminate self (TERM-SELF) request to the SSCP. The SNA Gateway sends TERM-SELF in response to an SNALU0$REQUEST_DISCONNECT call from the application.

**Figure 4–1 SNA Layers**




### 4.1.2 Presentation Services

Presentation services (PS) components convert user data into a format that is appropriate to the particular kind of data and to the application that receives it. In many LU-LU sessions, the PS components at both ends of the session communicate by means of function management headers (FMHs) located within the request unit. The OpenVMS application is responsible for providing all required presentation services, including the construction and interpretation of FMHs.

An example of presentation services is code translation. The OpenVMS application must perform all required ASCII/EBCDIC translations.

## 4.2 The Data Flow Control Layer

The Data Flow Control (DFC) layer, working in cooperation with the Transmission Control layer, ensures the proper flow of response units (RUs) between end users. Every session includes a data flow control component that is tailored to serve that session.

Peer DFC elements at either end of the session communicate by means of DFC RUs and by information included in the request/response headers attached to user data RUs.

### 4.2.1 Function Management Profiles

SNA defines subsets of commonly used data flow control functions in function management (FM) profiles. Each FM profile specifies a group of required and optional DFC functions for a session. The FM profile field in the BIND request indicates the subset of DFC functions that will be used during a session.

The SNA Gateway supports FM profiles 3, 4, 7, and 18.

### 4.2.2 Send/Receive Modes

DFC send/receive modes describe the way normal-flow RUs will travel between half-sessions. The BIND request specifies the type of send/receive mode to be used during the session. The API handles normal-flow RU traffic according to the specified mode.

SNA defines three send/receive modes:

- Half duplex flip-flop mode
- Half duplex contention mode
- Duplex mode

#### 4.2.2.1 Half Duplex Flip-Flop Mode

In half duplex flip-flop mode, the SLU and the PLU take turns being the sender of requests. At any moment, the LU currently designated as "requester" can shift this role to the other LU by sending a change direction indicator (CDI) in the request header attached to the RU. A DFC RU called SIGNAL (see Section 4.2.2.1) can be used by either LU to request the other LU to send the CDI. The CDI does not exist when the session is between brackets.

In LU type 0 sessions, the OpenVMS application is responsible for setting the CDI in the request header (via the *turn-retain* parameter in the TRANSMIT_ MESSAGE procedure) and sending the SIGNAL request as required for half duplex flip-flop mode.

#### 4.2.2.2 Half Duplex Contention Mode

In half duplex contention mode, either LU can send a request. If a request is received by the LU currently sending a request or request chain contention occurs. (See Section 4.2.3 for information about chains.) If this mode is used, the PLU and the SLU must agree at session establishment on who will win the contention. The contention resolution field in the BIND request specifies the winner. The API supports data exchanges in half duplex contention mode.

#### 4.2.2.3 Duplex Mode

In duplex mode, requests flow in both directions simultaneously. The flow of requests in one direction is independent of the flow of requests in the other. The API supports data exchanges in duplex mode.

### 4.2.3 Chains

Chaining is a technique used to send a series of RUs through the network as a single entity. RUs in a chain are either accepted or rejected as a unit by the receiving LU.

A chain can consist of a single RU or multiple RUs. All RUs in a chain are sent sequentially to the same destination. The chain is the basic unit of recovery. Chaining specifier fields in the BIND request specify whether or not the SLU or the PLU can send single- or multiple-element chains during the session.

The RH includes a chaining indicator field to indicate the position of each RU in the chain. There are three chaining indicators:

- Begin chain indicator (BCI) carried by the first RU
- End chain indicator (ECI) carried by the last RU
- Middle of chain indicator carried by all other RUs

By default, the API packages user data and commands into the appropriate RUs and transmits the RUs to the IBM host as a complete chain, setting the chaining indicators in the RH as required. If desired, the OpenVMS application can control the setting of the ECI (by means of the "more data" parameter in the TRANSMIT_MESSAGE procedure).

If a large enough buffer is provided, the API can receive a complete chain of RUs.

### 4.2.4 Response Types

SNA defines three types of responses:

- No response
- Exception response
- Definite response

Chain response specifier fields in the BIND request specify the type or types of response that can be used during the session. The form of the response requested field in the RH specifies the type of response required for a particular RU.

The OpenVMS application is responsible both for generating the proper responses to the requests it receives during a session and for providing sense data for negative responses, if necessary. The API adds the response header and chaining indicators required for transmitting the response to the PLU. The application is also responsible for interpreting received responses.

#### 4.2.4.1 No Response Chains

If the sender specifies "no response" in the RH, the receiver sends no response to any RU received in the chain. In a no response chain, every request unit in the chain carries a no response indicator. If the application responds when no response is required, it receives an error code.

#### 4.2.4.2 Exception Response Chains

If the sender specifies "exception response" in the RH, the receiver sends a response only in the event of an error. If no error occurs, the whole chain can be sent without acknowledgment from the receiver. In an exception response chain, every request unit in the chain carries an exception request indicator. The user must respond at the end of an exception response chain. The API will send the response if necessary and free resources.

#### 4.2.4.3 Definite Response Chains

If the sender specifies "definite response" in the RH for the last RU in a chain, the receiver must return a positive or negative response to that request. In a definite response chain, the last request unit in the chain carries a definite response indicator. All other requests carry an exception response indicator.

### 4.2.5 Request/Response Mode Protocols

If the session requires definite responses or exception responses for each chain, the communicating LUs also decide whether the sending LU must wait for responses before it sends additional requests. SNA request/response modes define the type and the number of chains that an LU can send before waiting for a response. There are two control modes:

- Immediate control mode
- Delayed control mode

In immediate control mode, each chain specifies a definite response, and only one chain requiring a response can be outstanding. In delayed control mode, multiple RU chains are allowed.

#### 4.2.5.1 Request Modes

If the session uses delayed control mode, the LUs must also decide on a request mode. SNA defines two request modes:

- Immediate request mode
- Delayed request mode

Request mode fields in the BIND request specify the request mode that the PLU and SLU can use during the session. In immediate request mode, only one chain requiring a definite response can be outstanding. In delayed request mode, more than one chain requiring a definite response can be outstanding.

The API rejects any user requests that violate the mode specified in the BIND agreement for the session.

#### 4.2.5.2 Response Modes

Each response carries an indicator–a sequence number or a unique ID–of the request to which it belongs. Response modes affect the order in which responses are sent. SNA defines two response modes:

- Immediate response mode
- Delayed response mode

In immediate response mode, the order in which requests are received determines the order in which responses are sent. In delayed response mode, the responses can be sent in any order.

### 4.2.6 Brackets

Brackets are indicators within the RH that define a series of RUs (requests and responses going in both directions) as a unit of work. A bracket is delimited by a begin bracket indicator (BBI) in the RH of the first request in the first chain, and an end bracket indicator (EBI) in the RH of the first request of the last chain.

At session establishment, the LUs agree on whether to use brackets for the session. If brackets are used, one LU is designated "first speaker." The first speaker can begin a bracket without asking permission of the other LU. The other LU becomes the "bidder." The bidder must ask permission to begin a bracket by sending a DFC bid request, as described in Section 4.2.7.5.

The BIND request contains the following fields that describe the way brackets will be used during a session:

- The bracket usage field specifies whether brackets will be used during the session.
- End bracket indicator (EBI) fields specify whether the SLU or the PLU can send the EBI.
- The bracket termination rule specifier field indicates whether bracket termination rule 1 or bracket termination rule 2 will be used for the session.
- The bracket first speaker field in the BIND request designates either the SLU or the PLU as first speaker.

Under bracket termination rule 1, the type of response required by the last-in-chain element (in a chain in which the EBI has been set) determines the way the bracket terminates. If the last-in-chain element requires a definite response, the bracket is terminated when the definite response is returned. If the last-in-chain element requires an exception response, the bracket ends when the last element of the chain is successfully received.

Under bracket termination rule 2, the bracket ends unconditionally upon receipt of the last-in-chain element of a chain in which the EBI indicator has been set.

In LU type 0 sessions involving the SNA Gateway, the API is responsible for setting the BBI if the session is using brackets. The OpenVMS application sets the EBI as required.

If the session is between brackets (the EBI was set in the last chain sent or received), the CDI does not exist and the LU may send.

### 4.2.7 Data Flow Control Requests

The DFC components in communicating LUs can generate their own requests to exchange control information. At session initiation, the LUs agree on the control requests that they can exchange.

The OpenVMS application is responsible for generating the DFC requests required for the session. The API adds the RH, the unique identifier or sequence number, and the chain indicators needed for sending the DFC request to the PLU. The API also determines whether to use normal or expedited flow. The OpenVMS application is responsible for processing DFC requests. The API ensures that the response is appropriate. These include the following:

- Cancel request
- Pause requests
  - Quiesce at end of chain
  - Quiesce complete
  - Release quiesce
- Cleanup requests
  - Chase
  - Shutdown
  - Shutdown complete
  - Request shutdown
- Signaling request
- Bracket requests
  - Bid request
  - Ready to receive request
- LU status request

#### 4.2.7.1 Cancel Request

The cancel (CANCEL) request can be sent by an LU to terminate a partially sent chain of RUs. CANCEL is sent only when a chain is in progress. The API first completes the receive with the *more-data* parameter set to true. The API then returns the CANCEL request on the next receive and discards all the remaining RUs in the chain. The user must discard all RUs received before the CANCEL.

FM profiles 3 and 4 specify SLU and PLU support for the CANCEL function.

#### 4.2.7.2 Pause Requests

For any number of reasons, the application or LU at one end of the session may ask the application or LU at the other end to pause for a while in its transmission of data. There are three pause requests:

- Quiesce at end of chain (QEC)
- Quiesce complete (QC)
- Release quiesce (RELQ)

Profile 4 specifies SLU and PLU support for QEC, QC, and RELQ. Profile 3 does not support pause requests. A logical place to pause is at the end of a chain of requests. To request the PLU to stop sending normal-flow requests at the end of the current chain, the SLU issues a QEC request on the normal flow.

When the PLU receives a QEC request, it returns a QC RU on the normal flow. QC is a normal-flow synchronizing request. It is the last normal flow request that the PLU sends until the SLU indicates that it is ready to resume receiving data.

To remove the quiesce condition imposed on the PLU, the SLU sends a RELQ request on the expedited flow. After receiving RELQ, the PLU can resume sending normal flow requests.

**Note**

QEC, QC, and RELQ affect normal-flow requests only. They do not affect expedited RUs or responses to normal-flow requests.

#### 4.2.7.3 Cleanup Requests

There are three cleanup requests:

- Chase (CHASE)
- Shutdown (SHUTD)
- Shutdown complete (SHUTC)
- Request shutdown (RSHUTD)

FM profile 3 and 4 specify SLU and PLU support for these cleanup requests.

The CHASE request tells the receiving LU to return all outstanding normal-flow responses and to send a response when this is done. Both the PLU and the SLU can send CHASE on the normal flow in preparation for a quiesce or shutdown to ensure that all preceding requests and responses have been processed before termination occurs.

SHUTD and SHUTC requests bring about an orderly termination of a session.

The PLU sends the SLU a SHUTD request on the expedited flow to say that the work is done, to instruct the SLU to complete end-of-session processing, and to quiesce when ready to end the session.

The SLU sends the PLU a SHUTC on the expedited flow to say that it has finished end-of-session processing and is entering the quiesce state.

The SLU sends the PLU an RSHUTD request on the normal flow to indicate that the work is done and to request either an UNBIND request or a CLEAR and UNBIND. (Note that, in spite of its name, RSHUTD does not request a SHUTD RU, but rather termination of the session.)

#### 4.2.7.4 Signal Request

The signal (SIGNAL) request is an expedited request that can be sent between half-sessions, regardless of the status of the normal flows. The request carries a 4-byte signal code: the first 2 bytes are the signal field and the last 2 bytes are the signal extension field. Profiles 3 and 4 specify SLU and PLU support for SIGNAL.

#### 4.2.7.5 Bracketing Requests

There are two bracketing requests:

- Bid (BID)
- Ready to receive (RTR)

If the SLU and the PLU agree to use bracketing, the normal flow BID request enables the half-session designated bidder to request permission to initiate a bracket.

The RTR request, issued on the normal flow by the first speaker, tells the bidder that it can now initiate a bracket. This RTR request is sent only after the PLU has a sent a negative response to a BID request.

#### 4.2.7.6 LU Status Request

The LU status (LUSTAT) request is typically used to report on failures and error recovery conditions for a local device of the LU. Either the PLU or the SLU can issue the LUSTAT request on the normal flow.

## 4.3 Transmission Control Layer

Half-session support for an LU includes a Transmission Control (TC) layer that contains the the following principal components:

- Connection point manager
- Session control

SNA defines subsets of TC functions in transmission subsystem (TS) profiles. Each TS profile specifies a group of required and optional TC functions for a session. The BIND request includes a TS profile field that indicates the subset of functions that can be used during the session. The BIND request also includes TS usage fields that specify the optional TC functions that will be used in the session.

### 4.3.1 Connection Point Manager

The connection point manager (CPMGR) is the half-session's interface to the common transmission network. The CPMGR performs the following functions:

- Constructs the RH for all outgoing RUs.
- Assigns a sequence number to each outgoing RU on the normal flow and interprets the sequence number for each incoming RU on the normal flow.
- Assigns a unique identifier for each outgoing RU on the expedited flow and interprets the ID for each incoming RU on the expedited flow. The OpenVMS application performs this function.
- Paces the sending of normal flow requests. The SNA Gateway performs this function.

#### 4.3.1.1 Building the Request/Response Header

The API is responsible for building a request/response header (RH) for each outgoing RU and for interpreting the RH header for each incoming RU. The OpenVMS application can control the setting of certain RH indicators. RH fields are described in Appendix C.

#### 4.3.1.2 Assigning Sequence Numbers and Unique Identifiers

The API is responsible for assigning a sequence number to each RU that it sends out on the normal flow and a unique identifier to each RU that it sends out on the expedited flow. The API passes to the OpenVMS application the sequence number or identifier for each received RU.

Each response has the same sequence number or unique ID as its associated request. This enables the receiver of the response to correlate responses with requests and to wait, if necessary, for the response to arrive.

#### 4.3.1.3 Pacing

At session establishment, each LU agrees not to send messages at a rate faster than the receiving LU can handle them. The agreement states that the sender will send up to a specified maximum of RUs. After that, it must wait for a go-ahead signal from the receiver before it can send more.

Pacing is handled by the API and the SNA Gateway. In most cases, the go-ahead signal is the pacing indicator in the RH of the next response. In the case of no-response exchanges, the receiver sends a special isolated pacing response. This is a message with no data, just the RH (with pacing indicator) and other headers.

The BIND request includes two pacing fields for the SLU: the send pacing field and the receive pacing field. The SLU send pacing field specifies the rate at which the SLU can send messages to the PLU. The SLU receive pacing field specifies the rate at which the PLU can send messages to the SLU. In SNA Gateway communications, the Gateway reads the SLU pacing fields in the BIND request.

For a session involving the SNA Gateway, the send pacing field must be zero filled, signifying that the SNA Gateway (representing the SLU) cannot ask the PLU to return pacing responses. The PLU is assumed to have enough buffer space to handle any transmission from the SLU. If this field is not zero filled, the SNA Gateway rejects the BIND request.

The value in the receive pacing field specifies the number of incoming RUs that can be buffered by the SNA Gateway. After the PLU transmits the number of RUs specified in this field, it requests a pacing response. If the SNA Gateway can accept more RUs, it returns the pacing response, and the PLU continues to send. If it cannot accept more RUs, it withholds the pacing response until such time as it can accept them.

### 4.3.2 Session Control

Session control functions for LU-LU sessions involve the following requests:

- Requests for managing the activation/deactivation of the session
  1. BIND
  2. UNBIND
- Requests for starting and clearing data traffic for an activated session
  1. Start data traffic (SDT)
  2. CLEAR

- Requests for assisting upper-level functions to resynchronize if an RU is lost or out of sequence
    1. Request recovery (RQR)
    2. Set and test sequence number (STSN)

#### 4.3.2.1 BIND and UNBIND Requests

To activate a session, the PLU sends the SLU a BIND request on the expedited flow. The BIND request consists of fields that the PLU sets to specify the rules and characteristics of the session. The SLU reads the BIND request and either accepts or rejects the session according to the characteristics specified.

In SNA Gateway communications, the IBM application sends the BIND request to the SNA Gateway. The SNA Gateway passes the BIND request to the OpenVMS application. The OpenVMS application inspects the BIND request and accepts or rejects the session. In SNA Gateway communications, the BIND request can be either negotiable or nonnegotiable. If negotiable, the OpenVMS application can accept a session proposed by a BIND request on the condition that certain parameters are changed. The OpenVMS application changes the values of these parameters in the BIND buffer and returns the entire buffer to IBM using the TRANSMIT_RESPONSE procedure.

To deactivate a session, the PLU sends an UNBIND (other than type 2) request. In SNA Gateway communications, the SNA Gateway receives the UNBIND request and notifies the OpenVMS application via the SNAEVT$K_TERM event code. At the time the OpenVMS application is notified, the session has already been terminated, and the OpenVMS application can only disconnect and release resources within the API.

The OpenVMS application is notified of the receipt of an UNBIND type 2 via the SNAEVT$K_UNBHLD notify event. To respond to an UNBIND type 2, the application must issue an SNALU0$REQUEST_RECONNECT. When the reconnect completes, the application must acknowledge the BIND and wait to receive the SDT (see Section 4.3.2.2).

#### 4.3.2.2 Starting and Clearing Data Traffic

To activate data traffic in a session, the PLU sends the SLU a start data traffic (SDT) request on the expedited flow. SDT starts the traffic flow for both directions.

To purge all RUs that may be traveling over the network, the PLU sends the SLU a CLEAR request on the expedited flow. CLEAR inhibits further normal flow traffic until the PLU issues an SDT.

Expedited data is received just as any other data message, through SNALU0$RECEIVE_MESSAGE. No other notification is given. The CLEAR RU is also received through SNALU0$RECEIVE_MESSAGE. After the applications +RSP's to the CLEAR, then the application is notified that the session has been "reset".

#### 4.3.2.3 Recovery/Resynchronization Functions

To direct data traffic recovery procedures, the SLU sends the PLU a request recovery (RQR) request on the expedited flow. The OpenVMS application can send the RQR request.

To resynchronize the sequence numbers at both ends of the session, the PLU sends a set and test sequence number (STSN) request on the expedited flow. The OpenVMS application is always the SLU and therefore never sends STSN.

The API takes care of the manipulation of sequence numbers for the OpenVMS application the application is simply informed that it is happening. The OpenVMS application, however, is responsible for responding to the STSN. It does this by issuing an SNALU0$TRANSMIT_RESPONSE with a *resp-type* of SNALU0$K_POSITIVE_RSP. The application is also responsible for setting and testing its own copy of sequence numbers, if it is maintaining one.

# 5 Procedure Calling Formats

This chapter describes the calling formats for the procedures provided by the Digital SNA Application Programming Interface (API) for OpenVMS product. These procedures include:

- SNALU0$EXAMINE_STATE
- SNALU0$RECEIVE_MESSAGE
- SNALU0$REQUEST_CONNECT
- SNALU0$REQUEST_RECONNECT
- SNALU0$REQUEST_DISCONNECT
- SNALU0$TRANSMIT_MESSAGE
- SNALU0$TRANSMIT_RESPONSE

Calls to the API procedures have the following general format:

*status=SNALU0$procedure-name[ W] (argument,....,[argument])*

where

| | |
|---|---|
| *status* | is a status code returned as a function value. |
| *procedure-name* | is the name of the API procedure you want to call. |
| *_W* | specifies a synchronous operation. |
| *( )* | delimits the argument list. |
| *[argument]* | indicates an optional argument. |
| *argument* | is a symbol containing information that the application passes to the API. The arguments associated with each of the procedures in this chapter use shorthand notation to describe the argument's characteristics. You can find a summary of these notations in Appendix A. |

You can pass arguments to the API 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, but of a special data structure called a character string descriptor.

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

## 5.1 SNALU0$EXAMINE_STATE

The SNALU0$EXAMINE_STATE procedure returns the following information about the port state for the specified session:

- General session state
- Transmit "turn"
- Bracket state
- Chain state
- Quiesce state
- Current value of the sequence number for transmits and receives on the normal and expedited flow

**Format:**

```
status.wlc.v=SNALU0$EXAMINE_STATE[_W](session-id.rlu.r,
                                      status-blk.wz.dx,
                                      [session-state.wbu.r],
                                      [bracket-state.wbu.r],
                                      [chain-send.wbu.r],
                                      [chain-receive.wbu.r],
                                      [turn-state.wbu.r],
                                      [quiesce-send.wbu.r],
                                      [quiesce-receive.wbu.r],
                                      [normal-send-seqno.wwu.r],
                                      [normal-receive-seqno.wwu.r],
                                      [expedited-seqno.wwu.r])
```

**Arguments:**

| | |
|---|---|
| *status* | When a procedure finishes execution, it returns a 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 as a function value. |
| *session-id* | A session identifier assigned at connect time. Passed by reference. |
| *status-blk* | A data structure to contain status information on completion of the procedure. Passed by descriptor. |
| *session-state* | A location to receive a value indicating the general session state (see Table 3–2). Passed by reference. |
| *bracket-state* | A location to receive a value indicating the bracket state for the session (see Table 3–2). Passed by reference. |
| *chain-send* | A location to receive a value indicating the chain send state for the session (see Table 3–2). |
| *chain-receive* | A location to receive a value indicating the chain receive state for the session (see Table 3–2). |
| *turn-state* | A location to receive a value indicating the transmit turn state for the session (see Table 3–2). Passed by reference. |
| *quiesce-send* | A location to receive a value indicating the quiesce send state for the session (see Table 3–2). Passed by reference. |
| *quiesce-receive* | A location to receive a value indicating the quiesce receive state for the session (see Table 3–2). |
| *normal-send-seqno* | A location to receive the current sequence number value for normal flow transmits. Passed by reference. |
| *normal-receive-seqno* | A location to receive the current sequence number value for normal flow receives. Passed by reference. |
| *expedited-seqno* | A location to receive the current sequence number value for expedited flow transmits. Passed by reference. |

The SNALU0$EXAMINE_STATE procedure can return the following status codes:

- SNALU0$_INVSESID
- SNALU0$_NORMAL
- SNALU0$_PARERR

## 5.2 SNALU0$RECEIVE_MESSAGE

The SNALU0$RECEIVE_MESSAGE procedure receives either a complete or partial chain of request units or a response unit from the PLU.

**Format:**

```
status.wlc.r=SNALU0$RECEIVE_MESSAGE[_W] (session-id.rlu.r,
                                         status-blk.wz.dx,
                                         buff.wx.dx,
                                         buff-size.wwu.r,
                                         req-ind.wbu.r,
                                         more-data.wbu.r,
                                         [msg-class.wbu.r],
                                         [msg-type.wbu.r],
                                         [flow.wbu.r],
                                         [alt-code.wbu.r],
                                         [beg-brack.wbu.r],
                                         [end-brack.wbu.r],
                                         [sense-inc.wbu.r],
                                         [resp-type.wbu.r],
                                         [end-data.wbu.r],
                                         [seq-num-first.ww.r],
                                         [seq-num-last.ww.r],
                                         [event-flag.rlu.r],
                                         [ast-addr.szem.r],
                                         [ast-par.rlu.r])
```

**Arguments:**

| | |
|---|---|
| *status* | When a procedure finishes execution, it returns a 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 as a function value. |
| *session-id* | A session identifier assigned at connect time. Passed by reference. |
| *status-blk* | A data structure to contain status information on completion of the procedure. Passed by descriptor. |
| *buff* | A buffer to contain either a complete or partial chain of request units or a response unit. The minimum size of the buffer is the RU size plus SNABUF$K_HDLEN. Passed by descriptor. |
| *buff-size* | A location to contain the actual length of the received data in bytes. Passed by reference. |

*req-ind*

A location to receive a symbol indicating whether the received RU is a request or a response:

- SNALU0$K_REQUEST indicates a request.
- SNALU0$K_RESPONSE indicates a response.

Passed by reference.

*more-data*

A location to receive a TRUE/FALSE flag specifying whether more data is to be received as part of the current chain.

- FALSE indicates that the last RU in the buffer carries the ECI. No more data is forthcoming for the current chain. The chain is complete.
- TRUE indicates that the last request unit in the buffer does not carry ECI. More data is forthcoming for the current chain. The chain is still open.

Passed by reference.

*msg-class*

A location to receive a value indicating the class of the message received (see Table 3–1). Passed by reference.

*msg-type*

A location to receive a value indicating the type of message received (see Table 3–1). Passed by reference.

*flow*

A location to receive a symbol indicating the flow on which the RU was received.

- SNALU0$K_NORMAL_FLOW indicates that the RU arrived on the normal flow.
- SNALU0$K_EXPEDITED_FLOW indicates that the RU arrived on the expedited flow.

Passed by reference.

*alt-code*

A location to receive a TRUE/FALSE flag indicating the presence of a character set other than EBCDIC within the user data.

- TRUE indicates that a non-EBCDIC character set is present.
- FALSE indicates that an alternate set is not present.

Passed by reference.

*beg-brack*

A location to receive a TRUE/FALSE flag indicating whether the first request unit in the chain carries the BBI.

- TRUE indicates that BBI is present.
- FALSE indicates that the BBI is not present.

Passed by reference.

*end-brack*

A location to receive a TRUE/FALSE flag indicating whether the first request unit in the chain carries the EBI.

- TRUE indicates that the EBI is present.
- FALSE indicates that the EBI is not present.

Passed by reference.

*sense-inc*

Valid for negative responses only. A location to receive a TRUE/FALSE flag indicating that the first four bytes of the negative response contain sense data.

- TRUE indicates that sense data is present.
- FALSE indicates that sense data is not present.

Passed by reference.

*resp-type*

A location to receive one of the following symbols specifying the type of response to be supplied to an RU that carries an ECI:

1. SNALU0$K_RSP_RQD1
2. SNALU0$K_RSP_RQD2
3. SNALU0$K_RSP_RQD3
4. SNALU0$K_RSP_RQE1
5. SNALU0$K_RSP_RQE2
6. SNALU0$K_RSP_RQE3
7. SNALU0$K_RSP_NONE

Passed by reference.

*end-data*

Valid for half duplex flip-flop mode only. A location to receive a TRUE/FALSE flag indicating the presence of the CDI in the request unit carrying the ECI. If the EBI is set (that is, if the session is between brackets), this parameter has no meaning (see Section 4.2.6).

- TRUE indicates that the CDI is present. The OpenVMS application can now send a message.
- FALSE indicates that the CDI is not present.

Passed by reference.

*seq-num-first*

A location to receive the sequence number assigned to the first RU in the received chain. Passed by reference.

*seq-num-last*

A location to receive the sequence number assigned to the last RU in the received RU. Passed by reference.

**Note**

If the receive buffer contains a chain consisting of a single data flow or session control RU, *seq-num-first* and *seq-num-last* both receive the unique ID assigned to the RU.

*event-flag*

An event flag to be set at completion. The default is event flag 0. Passed by reference.

*ast-addr*

The address of a user-written procedure called by the API upon completion. Passed by reference.

*ast-par*

An optional user-specified longword parameter to be passed to the user-written completion procedure. Passed by reference.

The SNALU0$RECEIVE_MESSAGE procedure can return the following status codes:

- SNALU0$_EVTCLR
- SNALU0$_GETLU0VM
- SNALU0$_INVBUF
- SNALU0$_INVSESID
- SNALU0$_NORMAL
- SNALU0$_PARERR
- SNALU0$_RCVBFSM

- SNALU0$_RCVFAIL
- SNALU0$_ILEFWT

## 5.3 SNALU0$REQUEST_CONNECT

The SNALU0$REQUEST_CONNECT procedure issues either an active or a passive request to establish a session between an OpenVMS application and an IBM application.

**Format:**

```
status.wlc.r=SNALU0$REQUEST_CONNECT[_W](session-id.wlu.r,
                                        status-blk.wz.dx,
                                        conn-type.rlu.r,
                                        [node-desc.rt.dx],
                                        [acc-name.rt.dx],
                                        [pu-name.rt.dx],
                                        [sess-addr.rlu.r],
                                        [applic-prog.rt.dx],
                                        [logon-mode.rt.dx],
                                        [user-id.rt.dx],
                                        [pass-word.rt.dx],
                                        [data.rt.dx],
                                        [notify-rtn.zem.r],
                                        [notify-parm.rlu.r],
                                        [notify-status.wz.dx]
                                        [bind-buf.wt.dx],
                                        [bind-len.wwu.r],
                                        [event-flag.rlu.r],
                                        [ast-addr.szem.r],
                                        [ast-par.rlu.r])
```

**Arguments:**

| | |
|---|---|
| *status* | When a procedure finishes execution, it returns a 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 as a function value. |
| *session-id* | A location to receive a unique session identifier that will be used in subsequent references to the session. Passed by reference. |
| *status-blk* | A data structure to contain status information on completion of the procedure. Passed by descriptor. |

*conn-type*

A value to specify the type of connection desired.

- SNALU0$K_ACTIVE indicates an active connection request.
- SNALU0$K_PASSIVE indicates a passive connection request.

Passed by reference.

*node-desc*

A Gateway DECnet node or TCP/IP host name string. For OpenVMS SNA, set this parameter to equal an ASCII 0. If the Gateway DECnet node name string is not supplied, the API assumes you are requesting a connection via OpenVMS SNA. Passed by descriptor.

*acc-name*

An access name associated with a list of default PLU access values. The maximum length is 8 characters. If you omit the access name, you must supply the IBM access information required in the parameter list. Passed by descriptor.

*pu-name*

A string defining the Gateway Physical Unit (PU) (for example, SNA-0) or the DECnet SNA or OpenVMS SNA used to establish the session with IBM. For Domain Gateway and Peer Server, use this string to supply the LU name in the SNA Gateway used for the session. The maximum length is 8 bytes. Passed by descriptor.

*sess-addr*

The number (in the range of 1 to 127) of the SLU over which the session is to take place. This information is not used, and do not supply it when connecting through a Domain Gateway or Peer Server. Passed by reference.

*applic-prog*

The PLU application to which you want to connect in the IBM host. The maximum length is 8 characters. Note that most IBM application names must be uppercase (for example, CICS). Passed by descriptor.

*logon-mode*

A logon mode name associated with a set of BIND request parameters for the session. The maximum length is 8 characters. Passed by descriptor.

*user-id*

A name identifying the user to the SSCP. The maximum length is 8 characters. Passed by descriptor.

*pass-word*

A password associated with the user ID. The maximum length is 8 characters. Passed by descriptor.

*data*

Optional user data. The maximum length is 128 characters. Passed by descriptor.

| | |
|---|---|
| *notify-rtn* | A notification procedure. This procedure is called by the API to notify the user application of network-related events. Although this parameter is optional, Digital strongly recommends that you provide a notify routine that can process the network events described in Section 3.6. Passed by reference. |
| *notify-parm* | An optional user-specified longword parameter to be passed to the notification procedure. Passed by reference. |
| *notify-status* | A data structure to contain information about an asynchronous event that has occurred during the session. Passed by descriptor. |
| *bind-buf* | A buffer to receive the BIND request image. The buffer should be at least SNABUF$K_LENGTH bytes in length. Passed by descriptor. If the application specifies a class S descriptor, the API copies the BIND request image into the buffer pointed to by the descriptor. If the application specifies a class D descriptor, the API fills in the descriptor to point to the BIND request image. The application is responsible for deallocating dynamic memory. |
| *bind-len* | A location to receive the length of the BIND request image. Passed by reference. |
| *event-flag* | An event flag to be set upon completion. The default is event flag 0. Passed by reference. |
| *ast-addr* | The address of a user-written procedure called by the API upon completion. Passed by reference. |
| *ast-par* | An optional user-specified longword parameter to be passed to the user-written completion procedure. Passed by reference. |

The SNALU0$REQUEST_CONNECT procedure can return the following status codes:

- SNALU0$_ACQLU
- SNALU0$_EVTCLR
- SNALU0$_GETLU0VM
- SNALU0$_INVBUF
- SNALU0$_NORMAL
- SNALU0$_NOSESN
- SNALU0$_NOTVECTSM
- SNALU0$_RCVBFSM
- SNALU0$_RCVFAIL

- SNALU0$_UNALEF
- SNALU0$_ILEFWT

## 5.4 SNALU0$REQUEST_RECONNECT

The SNALU0$REQUEST_RECONNECT procedure maintains a session over an LU whose associated general session port state is UNBIND_HOLD (PENDING). Upon completion of the procedure, the port state is BIND_PENDING, indicating that the OpenVMS application is awaiting a new BIND request from the PLU.

**Format:**

```
status.wlc.r=SNALU0$REQUEST_RECONNECT[_W] (session-id.rlu.r,
                                            status-blk.wz.dx,
                                            [bind-buf.wt.dx],
                                            [bind-len.wwu.r],
                                            [event-flag.rlu.r],
                                            [ast-addr.szem.r],
                                            [ast-par.rlu.r])
```

**Arguments:**

| | |
|---|---|
| *status* | When a procedure finishes execution, it returns a 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 as a function value. |
| *session-id* | A session identifier assigned at connect time. Passed by reference. |
| *status-blk* | A data structure to contain status information on completion of the procedure. Passed by descriptor. |
| *bind-buf* | A buffer to receive the BIND request image. Passed by descriptor. If the application specifies a class S descriptor, the API copies the BIND request image into the buffer pointed to by the descriptor. If the application specifies a class D descriptor, the API fills in the descriptor to point to the BIND request image. The application is responsible for deallocating dynamic memory. |
| *bind-len* | A location to receive the length of the BIND request image. Passed by reference. |
| *event-flag* | An event flag to be set upon completion of the procedure. The default is event flag 0. Passed by reference. |

| | |
|---|---|
| *ast-addr* | The address of a user-written procedure called by the API upon completion of the procedure. Passed by reference. |
| *ast-par* | An optional user-specified longword parameter to be passed to the user-written completion procedure. Passed by reference. |

The SNALU0$REQUEST_RECONNECT procedure can return the following status codes:

- SNALU0$_EVTCLR
- SNALU0$_GETLU0VM
- SNALU0$_INVBUF
- SNALU0$_INVSESID
- SNALU0$_NORMAL
- SNALU0$_PARERR
- SNALU0$_RCNFAIL
- SNALU0$_RCVBFSM
- SNALU0$_ILEFWT

## 5.5 SNALU0$REQUEST_DISCONNECT

The SNALU0$REQUEST_DISCONNECT procedure initiates an immediate termination of the specified session. The SNA Gateway sends an UNBIND request and waits for the response. When the procedure completes, all resources allocated to this session by the API are deallocated.

OpenVMS applications also have the ability to send a TERM-SELF request, as some IBM applications can not accept an UNBIND request. This is done by using the *disconnect-type* argument.

**Format:**

```
status.wlc.r=SNALU0$REQUEST_DISCONNECT[_W] (session-id.rlu.r,
                                            status-blk.wz.dx,
                                            [event-flag.rlu.r],
                                            [ast-addr.szem.r],
                                            [ast-par.rlu.r],
                                            [disconnect-type])
```

**Arguments:**

| | |
|---|---|
| *status* | When a procedure finishes execution, it returns a 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 as a function value. |
| *session-id* | A session identifier assigned at connect time. Passed by reference. |
| *status-blk* | A data structure to contain status information on completion of the procedure. Passed by descriptor. |
| *event-flag* | An event flag to be set upon completion of the procedure. The default is event flag 0. Passed by reference. |
| *ast-addr* | The address of a user-written procedure called by the API upon completion of the procedure. Passed by reference. |
| *ast-par* | An optional user-specified longword parameter to be passed to the user-written completion procedure. Passed by reference. |
| *disconnect-type* | Either SNALU0$K_UNBIND, or SNALU0$K_TERM-SELF. The default is SNALU0$K_UNBIND. Transmit a TERM-SELF or UNBIND and wait for the response passed by reference. |

The SNALU0$REQUEST_DISCONNECT procedure can return the following status codes:

- SNALU0$_DISCFAIL
- SNALU0$_EVTCLR
- SNALU0$_GETLU0VM
- SNALU0$_INVSESID
- SNALU0$_NORMAL
- SNALU0$_PARERR
- SNALU0$_ILEFWT

## 5.6 SNALU0$TRANSMIT_MESSAGE

The SNALU0$TRANSMIT_MESSAGE procedure packages user data into one or more RUs and sends the RUs to the PLU as a complete chain, the beginning of a chain, the middle of a chain, or the end of a chain, as required.

**Format:**

```
status.wlc.r=SNALU0$TRANSMIT_MESSAGE[_W] (session-id.rlu.r,
                                          status-blk.wz.dx,
                                          buff.rx.dx,
                                          buff-size.rwu.r,
                                          msg-class.rbu.r,
                                          [alt-code.rbu.r],
                                          [end-brack.rbu.r],
                                          [resp-type.rbu.r],
                                          [more-data.rbu.r],
                                          [turn-retain.rbu.r],
                                          [seq-num-first.wwu.r],
                                          [seq-num-last.wwu.r],
                                          [event-flag.rlu.r],
                                          [ast-addr.szem.r],
                                          [ast-par.rlu.r])
```

**Arguments:**

| | |
|---|---|
| *status* | When a procedure finishes execution, it returns a 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 as a function value. |
| *session-id* | A session identifier assigned at connect time. Passed by reference. |
| *status-blk* | A data structure to contain status information on completion of the procedure. Passed by descriptor. |
| *buff* | A buffer containing the data to be transmitted. Passed by descriptor. |
| *buff-size* | The size of the user buffer in bytes. Passed by reference. |
| *msg-class* | The class of the message to be sent (see Table 3–1). Passed by reference. |
| *alt-code* | A TRUE/FALSE flag indicating the presence of a character set other than EBCDIC within the user data.<br>• TRUE indicates that the character set contains a non-EBCDIC code.<br>• FALSE indicates that the character set contains EBCDIC code only.<br>The default is FALSE. Passed by reference. |

*end-brack*

A TRUE/FALSE flag indicating that the OpenVMS application wants to end the current bracket.

- TRUE indicates that the OpenVMS application wants to end the current bracket. The API sets the EBI in the first RU in the chain that contains ECI.
- FALSE indicates that the OpenVMS application wants the current bracket to remain open. The API does not set the EBI.

The default is FALSE. Passed by reference.

*resp-type*

A value to specify the type of response to be supplied to an RU that carries an ECI. This parameter can take the following values:

1. SNALU0$K_RSP_RQD1
2. SNALU0$K_RSP_RQD2
3. SNALU0$K_RSP_RQD3
4. SNALU0$K_RSP_RQE1
5. SNALU0$K_RSP_RQE2
6. SNALU0$K_RSP_RQE3
7. SNALU0$K_RSP_NONE

The default is RSP_NONE. Passed by reference.

*more-data*

A TRUE/FALSE flag telling the API whether to set the ECI on the last RU of the chain built as a result of this TRANSMIT call.

- If TRUE, the API does not set the ECI indicator in the last RU. More data is to follow in a subsequent TRANSMIT_MESSAGE call. The chain remains open. The *turn-retain* parameter must be set FALSE (see below).
- If FALSE, the API sets the ECI indicator in the last RU. No more data is to follow. The chain is complete.

The default is FALSE. Passed by reference.

| | |
|---|---|
| *turn-retain* | A TRUE/FALSE flag used to control the setting of the CDI.<br>• If TRUE, the API does not set the CDI indicator in the RU carrying the ECI. The IBM application remains the receiver. The API can send another chain or continue sending the current chain.<br>• If FALSE, the API sets the CDI indicator in the RU carrying the ECI. The IBM application becomes the sender. The API can receive a chain.<br>The default is FALSE. Passed by reference. |
| *seq-num-first* | A location to receive the sequence number assigned by the API to the first element of the transmitted chain. Passed by reference. |
| *seq-num-last* | A location to receive the sequence number assigned by the API to the last element of the transmitted chain. Passed by reference. |
| *event-flag* | An event flag to be set upon completion of the procedure. The default is event flag 0. Passed by reference. |
| *ast-addr* | The address of a user-written procedure called by the API upon completion of the procedure. Passed by reference. |
| *ast-par* | An optional user-specified longword parameter to be passed to the user-written completion procedure. Passed by reference. |

The SNALU0$TRANSMIT_MESSAGE procedure can return the following status codes:

- SNALU0$_EVTCLR
- SNALU0$_FREELU
- SNALU0$_GETLU0VM
- SNALU0$_INVBUF
- SNALU0$_INVSESID
- SNALU0$_NORMAL
- SNALU0$_PARERR
- SNALU0$_XMTFAIL
- SNALU0$_ILEFWT

## 5.7 SNALU0$TRANSMIT_RESPONSE

The SNALU0$TRANSMIT_RESPONSE procedure transmits a response to a specified RU or chain of RUs.

**Format:**

```
status.wlc.r=SNALU0$TRANSMIT_RESPONSE[_W] (session-id.rlu.r,
                                          status-blk.wz.dx,
                                          buff.rx.dx,
                                          buff-size.rwu.r,
                                          resp-type.rbu.r,
                                          [sense.rlu.r],
                                          [event-flag.rlu.r],
                                          [ast-addr.szem.r],
                                          [ast-par.rlu.r])
```

**Arguments:**

*status* — When a procedure finishes execution, it returns a 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 as a function value.

*session-id* — A session identifier assigned at connect time. Passed by reference.

*buff* — The buffer containing the RU associated with the positive response. Passed by descriptor.

*buff-size* — The size of the buffer. Passed by reference.

*resp-type* — The type of response (positive or negative) to be returned to the PLU. This parameter takes the following values:

- SNALU0$K_POSITIVE_RSP
- SNALU0$K_NEGATIVE_RSP

Passed by reference.

*sense* — 4 bytes of sense data to be returned to the PLU if a negative response is being transmitted. Passed by reference.

*event-flag* — An event flag to be set upon completion of the procedure. This must be in Digital format (least significant byte first) and not in IBM format. API will perform any manipulation needed prior to sending the data. Passed by reference.

*ast-addr* — The address of a user-written procedure called by the API upon completion of the procedure. Passed by reference.

| | |
|---|---|
| *ast-par* | An optional user-specified longword parameter to be passed to the user-written completion procedure. Passed by reference. |

The SNALU0$TRANSMIT_RESPONSE procedure can return the following status codes:

- SNALU0$_EVTCLR
- SNALU0$_GETLU0VM
- SNALU0$_INVBUF
- SNALU0$_INVSESID
- SNALU0$_NORMAL
- SNALU0$_PARERR
- SNALU0$_TONEGRSP
- SNALU0$_TOPOSRS
- SNALU0$_XMTFAIL
- SNALU0$_FREELU0VM
- SNALU0$_ILEFWT

# 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 E contains programming examples 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 OpenVMS VAX BASIC program.

```
$ BASIC MYPROG RET
```

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:SNALU0DEF'
```

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

```
$ MACRO/OBJECT=MYDIR:MYPROG SYS$LIBRARY:SNALU0DEF+SNALIBDEF+MYDIR:MYPROG
```

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 API 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 *Guide to OpenVMS Software Installation*. To use the shareable API procedures, you must specify a linker options file. This links your executable image with the shareable image SYS$SHARE:SNALU0SHR.EXE. For example:

```
$ LINK/EXE/MAP/MYPROG,SYS$INPUT:/OPTION
SYS$SHARE:SNALU0SHR/SHARE
CTRL/Z
$
```

The following example links your executable image with the debugger and the shareable image SYS$SHARE:SNALU0SHR.EXE. You must specify a linker options file.

```
$ LINK/EXE/MAP/DEBUG MYPROG,SYS$INPUT:/OPTION
SYS$SHARE:SNALU0SHR/SHARE
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 additional options, see the *OpenVMS Linker Reference Manual*. Also, refer to your language manual for additional information.

# A

# Summary Chart of Procedure Parameter Notation

This appendix summarizes the notation used to describe parameters in the Digital SNA Application Programming Interface for OpenVMS. For further information about notations and their definitions, see the "VAX Procedure Calling and Condition Handling Standard" in the *Introduction to OpenVMS System Routines*.

The following format illustrates the location of the notation in the parameter:

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

where

1. <Name> is a mnemonic for the procedure.
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.

```
        <access type>

c     Call after stack unwind
f     Function call (before return)
j     JMP after unwind
m     Modify access
r     Read-only access
s     Call without stack unwinding
w     Write-only access
```

<data type>

```
a     Virtual address
adt   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
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)
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

# BIND Request Parameters

This appendix shows in Table B–1, the symbolic codes used for the BIND Request.

**Table B–1 Symbolic Codes for the BIND Request**

| Byte Number | Symbolic Code | Description |
|---|---|---|
| Byte 0 | SNABND$B_B0 | |
| | SNABND$B_CMD | BIND command |
| Byte 1 | SNABND$B_B1 | |
| | SNABND$B_BFMT | BIND format |
| Byte 2 | SNABND$B_B2 | |
| | SNABND$B_FMP | FM profile |
| Byte 3 | SNABND$B_B3 | |
| | SNABND$B_TSP | TS profile |
| Byte 4 | SNABND$B_B4 | Primary LU protocols |
| | SNABND$V_P_CHU | Chaining use |
| | SNABND$V_P_RMS | Request mode selection |
| | SNABND$V_P_CHR | Chaining responses |
| | SNABND$V_PH_COM | 2-phase commit for sync point |
| | SNABND$V_R0 | Reserved |

(continued on next page)

**Table B–1 (Cont.) Symbolic Codes for the BIND Request**

| Byte Number | Symbolic Code | Description |
|---|---|---|
| | SNABND$V_P_CMP | Compression indicator |
| | SNABND$V_P_SEB | Send end bracket indicator |
| Byte 5 | SNABND$B_B5 | Secondary LU protocols |
| | SNABND$V_S_CHU | COLUMN {Chaining use |
| | SNABND$V_S_RMS | Request mode selection |
| | SNABND$V_S_CHR | Chaining responses |
| | SNABND$V_S_TPC | 2-phase commit for sync point |
| | SNABND$V_R1 | Reserved |
| | SNABND$V_S_CMP | Compression indicator |
| | SNABND$V_S_SEB | Send end bracket indicator |
| Byte 6 | SNABND$B_B6 | Common protocols |
| | SNABND$V_SEG | Session segmenting |
| | SNABND$V_FMU | FM header usage |
| | SNABND$V_BIS | BIS sent |
| | SNABND$V_BRU | Bracket usage |
| | SNABND$V_BTP | Bracket termination protocol |
| | SNABND$V_ACS | Alternate code selection |
| | SNABND$V_SNA | Sequence number availability |
| | SNABND$V_BRQ | BIND response queue capability |
| Byte 7 | SNABND$B_B7 | Common protocols |
| | SNABND$V_NFM | Normal flow mode |
| | SNABND$V_RCR | Recovery responsibility |
| | SNABND$V_BFS | Bracket first speaker |
| | SNABND$V_ALPI | Alternate code processing ASCII-7 or -8 |
| | SNABND$V_CVI | Control vectors included |

(continued on next page)

**Table B–1 (Cont.) Symbolic Codes for the BIND Request**

| Byte Number | Symbolic Code | Description |
|---|---|---|
| | SNABND$V_CNR | Contention resolution |
| Byte 8 | SNABND$B_B8 | Secondary send pacing |
| | SNABND$V_SSI | Secondary CPMGRs staging indicator |
| | SNABND$V_R2 | Reserved |
| | SNABND$V_SSW | Secondary CPMGRs send window size |
| Byte 9 | SNABND$B_B9 | Secondary receive pacing |
| | SNABND$V_R3 | Reserved |
| | SNABND$V_SRW | Secondary CPMGRs receive window size |
| Byte 10 | SNABND$B_B10 | Maximum secondary RU size |
| | SNABND$V_S_MRU | Maximum RU size sent by SLU |
| | SNABND$V_S_MAN | Mantissa |
| | SNABND$V_S_EXP | Exponent |
| Byte 11 | SNABND$B_B11 | Maximum primary RU size |
| | SNABND$V_P_MRU | Maximum RU size sent by PLU |
| | SNABND$V_P_MAN | Mantissa |
| | SNABND$V_P_EXP | Exponent |
| Byte 12 | SNABND$B_B12 | Primary send pacing |
| | SNABND$V_PSI | Primary CPMGRs staging indicator |
| | SNABND$V_R4 | Reserved |
| | SNABND$V_PSW | Primary CPMGRs send window size |

(continued on next page)

**Table B–1 (Cont.) Symbolic Codes for the BIND Request**

| Byte Number | Symbolic Code | Description |
|---|---|---|
| Byte 13 | SNABND$B_B13 | Primary receive pacing |
| | SNABND$V_R5 | Reserved |
| | SNABND$V_PRW | Primary CPMGRs receive window size |
| Byte 14 | SNABND$B_B14 | PS profile |
| | SNABND$V_PUF | PS usage field |
| | SNABND$V_LUT | LU type |
| Bytes 15-27 | SNABND$B_B*xx* | Reserved (*xx* is a byte number from 15-27) |

**Table B–2 A BIND Request**

| Field | Location (IBM position) | Values and Meanings |
|---|---|---|
| RU identity | Byte 0 | An entry of x'31' identifies this RU as a BIND request. Mandatory field; if the values entered in this field are unacceptable, the SNA Gateway will reject the BIND. |
| BIND type/format | Byte 1 | x'01' or x'00' are the only valid entries. x'01' means that the bind is not negotiable, and x'00' means that the field is negotiable. Mandatory field; if the values entered in this field are unacceptable, the SNA Gateway will reject the BIND. |
| FM profile | Byte 2 | x'02', x'03', x'04', x'07', or x'12' depending upon the range of SNA commands you want to use in a session. Profile 04 permits you to use more SNA commands. Mandatory field; if the values entered in this field are unacceptable, the SNA Gateway will reject the BIND. If the values are unacceptable to the application, the application must reject them. |
| TS profile | Byte 3 | x'02', x'03', x'04', or x'07' depending upon the level of transmission recovery you want to make available to the session. Mandatory field; if the values entered in this field are unacceptable, the SNA Gateway will reject the BIND. If the values are unacceptable to the application, the application must reject them. |
| PLU protocols | Byte 4 | |
| Chaining specifier for PLU | Byte 4 bit 0 | 0 means that the PLU can send single-element chains only. 1 means that the PLU can send single- or multiple-element chains. |

(continued on next page)

**Table B–2 (Cont.) A BIND Request**

| Field | Location (IBM position) | Values and Meanings |
|---|---|---|
| Request mode specifier | Byte 4 bit 1 | 0 means that the session operates in immediate request mode. The PLU will not transmit a request if a definite response is outstanding. |
| | | 1 means that the session operates in delayed request mode. More than one definite response can be outstanding. |
| Chaining response specifier | Byte 4 bits 2,3 | 00 means the PLU cannot request a response. |
| | | 01 means that the PLU can request exception responses only. |
| | | 10 means that the PLU can request definite responses only. |
| | | 11 means that the PLU can request definite or exception responses. |
| Sync point specifier | Byte 4 bit 4 | For TS profile 4. 0 means that the 2-phase commit is not supported. 1 means that the 2-phase commit is supported. |
| (Reserved field) | Byte 4 bit 5 | Reserved. |
| Compression indicator for PLU | Byte 4 bit 6 | 0 means that the PLU cannot send compressed data. |
| | | 1 means that the PLU can send compressed data. The SNA Gateway does not restructure data formats. User-created data is passed directly to the cooperating application, which in turn must perform whatever conversion is necessary. |
| End bracket indicator for PLU | Byte 4 bit 7 | 0 means that the PLU cannot send the EBI. |
| | | 1 means that the PLU can send the EBI. If brackets are not allowed for the session (byte 6, bit 2 is 0), this bit must not be set. |

(continued on next page)

**Table B–2 (Cont.) A BIND Request**

| Field | Location (IBM position) | Values and Meanings |
|---|---|---|
| SLU protocols | Byte 5 | |
| Chaining specifier for SLU | Byte 5 bit 0 | 0 means that the SLU can send single-element chains only.<br>1 means that the SLU can send single- or multiple-element chains. |
| Request mode specifier for SLU | Byte 5 bit 1 | 0 means that the session operates in immediate request mode. The SLU will not transmit a request if there is a definite response outstanding.<br>1 means that the session operates in delayed request mode. More than one definite response can be outstanding. |
| Chaining response specifier for SLU | Byte 5 bits 2,3 | 00 means that the SLU cannot request a response.<br>01 means that the SLU can request exception responses only.<br>10 means that the SLU can request definite responses only.<br>11 means the SLU can request either definite or exception responses. |
| Sync point specifier | Byte 5 bit 4 | For TS profile 4. 0 means that 2-phase commit is not supported. 1 means that 2-phase commit is supported. |
| (Reserved field) | Byte 5 bit 5 | Reserved. |
| Compression indicator for SLU | Byte 5 bit 6 | 0 means that the SLU cannot send compressed data.<br>1 means that the SLU can send compressed data. The SNA Gateway does not attempt to restructure data. The cooperating applications must perform whatever conversion is necessary. |

(continued on next page)

**Table B–2 (Cont.) A BIND Request**

| Field | Location (IBM position) | Values and Meanings |
|---|---|---|
| End bracket indicator for SLU | Byte 5 bit 7 | 0 means that the SLU cannot send the EBI.<br>1 means that the SLU can send the EBI. |
| Common protocols | Bytes 6,7 | |
| (Reserved field) | Byte 6 bit 0 | Reserved. |
| FMH usage indicator | Byte 6 bit 1 | 0 means that function management headers (FMHs) cannot be exchanged in this session.<br>1 means that FMHs can be exchanged in this session. |
| Bracket usage indicator | Byte 6 bit 2 | 0 means that the cooperating applications agree that bracket protocol will not be used in this session. If this bit is set to 0, the following bits in the BIND must also be 0: byte 4, bit 7; byte 5, bit 7; byte 6, bit 3.<br>1 means that the cooperating applications agree that bracket protocol will be used in this session. If this bit is set to 1, byte 4, bit 7 or byte 5, bit 7, or both, must be set to 1. Byte 7, bit 3 must be set to 0; byte 7, bits 0-1 cannot be set to 00; bytes 4 and 5, bits 2-3 cannot be set to 00. |

(continued on next page)

**Table B–2 (Cont.) A BIND Request**

| Field | Location (IBM position) | Values and Meanings |
|---|---|---|
| Bracket termination rule specifier | Byte 6 bit 3 | 0 means that bracket termination rule 2 is used: the bracket ends unconditionally upon receipt of the last-in-chain element of a chain in which the EBI has been set.<br><br>1 means that bracket termination rule 1 is used: termination of a bracket depends upon the type of response required by the last-in-chain element of a chain in which the EBI has been set. If response type = definite, the bracket is terminated when the definite response is returned; if response type = exception, the bracket ends when the last element of that chain is successfully received.<br><br>When both the BBI and the EBI are set in the first-in-chain RH, the bracket ends unconditionally when the last RU for the chain is sent. |
| Alternate code indicator | Byte 6 bit 4 | 0 means that the cooperating applications must use a standard code (EBCDIC).<br><br>1 means that the cooperating applications can use an alternate code.<br><br>The code selection indicator in the RHs transmitted by the cooperating applications should be consistent. |
| Sequence number availability | Byte 6 bit 5 | For TS profile 4. 0 means that sequence numbers are not available. 1 means that sequence numbers are available. |
| (Reserved fields) | Byte 6 bit 6 | Reserved. |

(continued on next page)

**Table B–2 (Cont.) A BIND Request**

| Field | Location (IBM position) | Values and Meanings |
|---|---|---|
| BIND queuing indicator | Byte 6 bit 7 | 0 means that the BIND cannot be queued.<br>1 means that the BIND receiver can queue the BIND. |
| Normal flow mode specifier | Byte 7 bits 0,1 | 00 means that the cooperating applications interact in full duplex transmission mode. Byte 6, bit 2 must be set to 0, indicating that the cooperating applications agree that bracket protocol is not used in this session.<br>01 means that the session operates in half duplex (HDX) contention mode. If the value entered in this field is unacceptable, the SNA Gateway will reject the BIND.<br>If the cooperating applications agree that bracket protocol will not be used in the session, the following settings will be observed in the BIND:<br>Byte 4, bit 7 = 0: PLU cannot send EBI.<br>Byte 5, bit 7 = 0: SLU cannot send EBI.<br>Byte 6, bit 2 = 0: Session is unbracketed.<br>Byte 6, bit 3 = 0: Bracket termination rule 2 is used.<br>Byte 7, bit 7 = 0: SLU wins contention. |

(continued on next page)

**Table B–2 (Cont.) A BIND Request**

| Field | Location (IBM position) | Values and Meanings |
|---|---|---|
| | | If the cooperating applications agree that bracket protocol will be used in the session, the following settings will be observed in the BIND:<br>Byte 4, bit 7 = 1: PLU can send the EBI, *or*<br>Byte 5, bit 7 = 1: SLU can send the EBI.<br>Byte 4, bit 2 or 3: Either or both must be set to 1.<br>Byte 5, bit 2 or 3: Either or both must be set to 1.<br>Byte 6, bit 2 = 1: Session uses bracket protocol.<br>Byte 7, bit 3 = 1: SLU is bracket first speaker.<br>Byte 7, bit 7 = 0: SLU wins contention. |
| | | 10 means that the cooperating applications agree to interact in half duplex flip-flop (HDX-FF) transmission mode. The applications are written so that one is in the send state and in control of the change direction indicator (CDI), while the other is in receive state until it gains control of the CDI. |
| | | If byte 7, bit 7 is set to 0, the SLU is in send state when the session opens; if byte 7, bit 7 is set to 1, the PLU is in send state when the session opens. The application in receive state can request the CDI by issuing the SIGNAL request with a code of x'00000001' to the application in send state. The protocol to be observed in exchanging the CDI must be defined by the user. |

(continued on next page)

**Table B–2 (Cont.) A BIND Request**

| Field | Location (IBM position) | Values and Meanings | |
|---|---|---|---|
| | | If the cooperating applications agree that the HDX-FF session will not use bracket protocol, the following settings will be observed in the BIND: | |
| | | Byte 4, bit 7 = 0: | The PLU cannot send the EBI. |
| | | Byte 5, bit 7 = 0: | The SLU sends the EBI. |
| | | Byte 6, bit 2 = 0: | The cooperating applications agree that brackets will not be used. |
| | | Byte 6, bit 3 = 0: | Bracket termination rule 2 is used; the bracket ends unconditionally when the last RU of an EBI chain is received. |
| | | Byte 7, bit 7 = 0 or 1: | Either the PLU or the SLU must open the session in the send state and in control of the CDI. |
| | | If the cooperating applications agree that the HDX-FF session will use bracket protocol, the following settings will be observed in the BIND: | |
| | | Byte 4, bit 2 or 3 or both, must be set to 1. The PLU must specify a response type. | |
| | | Byte 4, bit 7, or byte 5, bit 7, or both = 1: | Either or both applications can send the EBI. |

(continued on next page)

**Table B–2 (Cont.) A BIND Request**

| Field | Location (IBM position) | Values and Meanings | |
|---|---|---|---|
| | | Byte 5, bit 2 or 3: | Either or both must be set to 1. The SLU must specify a response type. |
| | | Byte 6, bit 2 = 1: | The cooperating applications agree that brackets will be used in the session. |
| | | Byte 7, bit 7: | The setting of this bit has no effect on a bracketed HDX-FF session. |
| Recovery responsibility | Byte 7 bit 2 | 0 means that the SLU goes into a receive state when it receives a negative response. The PLU is responsible for attempting recovery. | |
| | | 1 means that the sending application remains in send state when a negative response is returned. Both applications employ user-defined protocols in attempting recovery. | |

(continued on next page)

**Table B–2 (Cont.) A BIND Request**

| Field | Location (IBM position) | Values and Meanings |
|---|---|---|
| Bracket first speaker | Byte 7 bit 3 | 0 means that the SLU is bracket first speaker. If a contention condition arises (i.e., if both applications attempt to send simultaneously), the application will first return a negative response to the PLU and then begin a bracket. The negative response returned by the SLU must contain sense data indicating whether it will accept a BID request (a request to begin a bracket) from the PLU when the current bracket is ended. If yes, the SLU must send a ready to receive (RTR) request to the PLU when the bracket ends. After receiving the RTR, the PLU can send a BID. |
| | | 1 means that the PLU is bracket first speaker. If this field is set to 1, a session must open with the SLU in the receive state. Also, the PLU is bracket first speaker. |
| Alternate code processing identifier | Byte 7 bits 4-5 | Byte 6, bit 4 must be set to 1 as this indicates that an alternate code is used. |
| | | 00 means that the application processes alternate code FMD RUs as ASCII-7. |
| | | 01 means that the application processes alternate code FMD RUs as ASCII-8. |
| Control vector | Byte 7 bit 6 | 0 means that control vectors are not included after the SLU name. |
| | | 1 means that control vectors are included after the SLU name. |

(continued on next page)

**Table B–2 (Cont.) A BIND Request**

| Field | Location (IBM position) | Values and Meanings |
|---|---|---|
| Contention resolution | Byte 7 bit 7 | 0 means that the SLU always wins contention. If the session is HDX contention, this bit must be set to 0.<br>1 means that the PLU wins contention. If the session is HDX-FF, unbracketed, the SLU must open in receive state. |
| Secondary pacing, SLU to PLU | Byte 8 | This field is zero-filled, meaning that the SNA Gateway cannot ask the PLU to return pacing responses. Digital recommends that you use a value of 0, but you can use values from 0 to 3F. It is assumed that the PLU buffering capability is adequate for any transmission from the SLU. Mandatory field. If the value entered in this field is unacceptable, the SNA Gateway will reject the BIND. |
| Secondary pacing, PLU to SLU | Byte 9 | The value entered in this byte defines the number of RUs that can be buffered by the SNA Gateway. After the PLU transmits the number of RUs specified in this byte, it requests a pacing response. If the SNA Gateway can accept more RUs, it will return the pacing response, enabling the PLU to continue sending. If the SNA Gateway does not return a pacing response, transmission will halt and then resume when the response is returned. If the value entered in this field is unacceptable, the SNA Gateway will reject the BIND. |

(continued on next page)

**Table B–2 (Cont.) A BIND Request**

| Field | Location (IBM position) | Values and Meanings |
|---|---|---|
| RU size, SLU to PLU | Byte 10 | Specifies the maximum length of RUs that can be transmitted from the SLU to the PLU. The value entered is not checked by the SNA Gateway. For SNA Gateway V1.2 or earlier, this value must be less than or equal to 256. For later versions of the SNA Gateway, you can use any value up to 4096. |
| RU size, PLU to SLU | Byte 11 | Specifies the maximum length of RUs that can be transmitted from PLU to the SLU. The value entered is not checked by the SNA Gateway |
| Primary pacing | Bytes 12,13 | |
| Primary pacing stages | Byte 12 bit 0 | 0 means that the primary send window size and the secondary receive window size are for two-stage pacing.<br>1 means that the primary send window size and the secondary receive window size are for one-stage pacing. |
| (Reserved bit) | Byte 12 bit 1 | Reserved. |
| Primary send pacing | Byte 12 bits 2-7 | The binary values entered here indicate the primary send window size for session-level pacing. |
| (Reserved bits) | Byte 13 bits 0-1 | Reserved. |
| Primary receive pacing | Byte 13 bits 2-7 | The binary values entered here indicate the primary receive window size for session-level pacing. |

(continued on next page)

**Table B–2 (Cont.) A BIND Request**

| Field | Location (IBM position) | Values and Meanings |
|---|---|---|
| SLU type | Byte 14 | x'00', x'01', x'02', or x'03' are the only valid entries. This value specifies the LU type used. |
| Session-related values | Bytes 15-27 | Values that can be entered here depend upon individual session parameters. (See SNA documentation.) |

# C
# The Request Response Header

A definition of the request/response header is included in the SNALIBDEF file for each language. Table C–1 provides a list of symbolic codes you can use to reference each bit of the request/response header. Figure C–1 illustrates the position of each bit in the request/response header.

**Table C–1 Symbolic Codes for the Request/Response Header**

| Byte Number | Symbolic Code | Description |
|---|---|---|
| Byte 0 | SNARH$V_ECI | End chain indicator |
| | SNARH$V_BCI | Begin chain indicator |
| | SNARH$V_SDI | Sense data included indicator |
| | SNARH$V_FI | Format indicator |
| | SNARH$V_RE0 | Reserved field |
| | SNARH$V_RUC | Request/response unit category |
| | SNARH$V_RRI | Request/response indicator |
| Byte 1 | SNARH$V_PI | Pacing indicator |
| | SNARH$V_QRI | Queued response indicator |
| | SNARH$V_RE1 | Reserved field |
| | SNARH$V_ER1 | Exception response indicator (request only) |
| | SNARH$V_RTI | Response type indicator (response only) |
| | SNARH$V_DR2I | Definite response 2 indicator |

(continued on next page)

**Table C–1 (Cont.) Symbolic Codes for the Request/Response Header**

| Byte Number | Symbolic Code | Description |
|---|---|---|
| | SNARH$V_RE2 | Reserved field |
| | SNARH$V_DR1I | Definite response 1 indicator |
| Byte 2 | SNARH$V_RE3 | Reserved field |
| | SNARH$V_PDI | Padded data indicator |
| | SNARH$V_EDI | Enciphered data indicator |
| | SNARH$V_CSI | Code selection indicator |
| | SNARH$V_RE4 | Reserved field |
| | SNARH$V_CDI | Change direction indicator |
| | SNARH$V_EBI | End bracket indicator |
| | SNARH$V_BBI | Begin bracket indicator |

**Figure C–1 Request/Response Header**

| 0* | | | | | | | 7 | |
|---|---|---|---|---|---|---|---|---|
| Request Response Indicator (RRI) | **RU Category** | | **Reserved** | Format Indicator (FI) | Sense Data Indicator (SDI) | Begin Chain Indicator (BCI) | End Chain Indicator (ECI) | Byte 0 |
| Definite Response 1 Indicator (DR1I) | **Reserved** | Definite Response 2 Indicator (DR2I) | Exception** Response Indicator (ERI) | **Reserved** | | Queued Response Indicator (QRI) | Pacing Indicator (PI) | Byte 1 |
| Begin Bracket Indicator (BBI) | End Bracket Indicator (EBI) | Change Bracket Indicator (CBI) | **Reserved** | Code Selection Indicator (CSI) | Enciphered Data Indicator (EDI) | Padded Data Indicator (PDI) | **Reserved** | Byte 2 |

* Bits numbered according to IBM.
** Request header only. In a response header, this bit is the response type indicator (RTI).

LKG–0165–93R

**Note**

Application programs do not deal with the following session control commands:

- Activate physical unit
- Activate logical unit
- Deactivate physical unit
- Deactivate logical unit

Asterisks mark these commands in Table C–2.

**Table C–2 Request Header**

| Field | Location | Values and Functions |
|---|---|---|
| Request indicator | Byte 0 bit 0 | 0 is the only valid value for this field. It means that the accompanying message is a request. |
| RH category | Byte 0 bits 1,2 | 00 means that function management data follows.<br>01 means that network control data follows. This data is not valid for LU-LU sessions.<br>10 means that DFC data follows. This RH is valid for the following RUs:<br>Logical unit status (LUSTAT)<br>Ready to receive BBI (RTR)<br>Quiesce at end of chain (QEC)<br>Quiesce complete (QC)<br>Release quiesce (RELQ)<br>Cancel chain (CANCEL)<br>Chase (CHASE)<br>Shutdown (SHUTD)<br>Shutdown complete (SHUTC)<br>Request shutdown (RSHUTD)<br>Bid for begin bracket (BID) |

(continued on next page)

**Table C–2 (Cont.) Request Header**

| Field | Location | Values and Functions |
|---|---|---|
| | | 11 means that session control (SC) data follows. This RH is valid for the following RUs:<br><br>* Activate physical unit (ACTPU)<br>* Activate logical unit (ACTLU)<br>Bind session (BIND)<br>Start data traffic (SDT)<br>Clear (CLEAR)<br>Set and test sequence numbers (STSN)<br>Request recovery (RQR)<br>Unbind session (UNBIND)<br>* Deactivate logical unit (DACTLU)<br>* Deactivate physical unit (DACTPU) |
| (Reserved field) | Byte 0 bit 3 | Reserved. |
| Format indicator | Byte 0 bit 4 | 0 means that RU is unformatted.<br>1 means that the RU contains an FMH.<br>This field is set to 1 for all DFC, SC, and NC commands. |
| Sense data included | Byte 0 bit 5 | In a request, this bit is always set to 0, meaning that sense data is not included in the accompanying RU. |
| Chaining indicator | Byte 0 bits 6,7 | 10 means that associated message is first in chain.<br>00 means that associated message is middle of chain.<br>01 means that associated message is last in chain. Cancel chain is always last in chain.<br>11 means that associated message is only in chain. All commands listed, except cancel chain, are only in chain. |

(continued on next page)

**Table C–2 (Cont.) Request Header**

| Field | Location | Values and Functions |
|---|---|---|
| Form of response indicated | Byte 1<br>bits 0,2,3;<br>bit 1*x* is<br>reserved | 1*x*00 means that a definite response, either positive or negative, is requested. All SC and DFC commands listed request a definite response by means of this bit configuration.<br>0*x*10 or 1*x*10 can also be used to request a definite response.<br>1*x*01, or 0*x*11, or 1*x*11 can be used to request an exception response. An exception response is returned only if it is negative.<br>0*x*00 means that a response is not requested. |
| (Reserved field) | Byte 1<br>bits 4-6 | Reserved. |
| Pacing indicator | Byte 1<br>bit 7 | 0 means that a pacing response is not requested. Expedited SC and DFC commands, listed below, do not request a pacing response:<br>Start data traffic (SDT)<br>Clear (CLEAR)<br>Set and test sequence number (STSN)<br>Request recovery (RQR)<br>Request shutdown (RSHUTD)<br>Shutdown (SHUTD)<br>Shutdown complete (SHUTC)<br>Quiesce at end of chain (QEC)<br>Release quiesce (RELQ)<br>1 means that a pacing response is requested.<br>All normal flow commands can use either a 1 or a 0 setting in this field. The SNA Gateway and the API determine the value of this field. |

(continued on next page)

**Table C–2 (Cont.) Request Header**

| Field | Location | Values and Functions |
|---|---|---|
| Begin bracket indicator | Byte 2 bit 0 | 0 means that this transmission does not begin a bracket.<br>1 means that this transmission does begin a bracket.<br>All commands listed have a 0 setting for this field. |
| End bracket indicator | Byte 2 bit 1 | 0 means that this transmission does not end a bracket.<br>1 means that this transmission does end a bracket.<br>All commands listed have a 0 setting for this field. |
| Change direction indicator | Byte 2 bit 2 | Used only in HDX-FF exchange mode on the normal flow. If bit is set, the receiver of the message is driven to the send state, and the sender is driven to receive state. |
| (Reserved bit) | Byte 2 bit 3 | |
| Code selection indicator | Byte 2 bit 4 | 0 means that the applications agree to use EBCDIC code.<br>1 means that the applications agree to use an alternate code.<br>All commands listed use a 0 setting in this field. |

**Table C–3 Response Header**

| Field | Location | Values and Functions |
|---|---|---|
| Response indicator | Byte 0 bit 0 | 1 is the only valid entry for this field. It means that the message is a response. |
| RH category | Byte 0 bits 1,2 | The entry in this field must echo the entry in the corresponding field of the RH. |

(continued on next page)

**Table C–3 (Cont.) Response Header**

| Field | Location | Values and Functions |
|---|---|---|
| (Reserved field) | Byte 0 bit 3 | Reserved. |
| Format indicator | Byte 0 bit 4 | Entry in this field must echo the entry in the corresponding field of the RH. |
| Sense data included | Byte 0 bit 5 | 0 means that the accompanying response does not carry any sense data. If this field is set to 0, then either definite response 1 indicator (DR1I) or definite response 2 indicator (DR2I) in the RH must have been set to 1, and response type indicator (RTI) in this RH must be set to 0, indicating a positive response.<br>1 means that the accompanying response carries 4 bytes of sense data. If this field is set to 1, then DR1I and DR2I in the RH must have been set to 1, and RTI in this RH must be set to 1, indicating a negative response. |
| Chaining indicators | Byte 0 bits 6,7 | 11 is the only valid setting for these fields. Indicates that the response is only in chain. |
| Form of response requested | Byte 1 bits 0-2 | Entries in these fields must echo entries in the corresponding fields of the RH.<br>Bit 1*x* and bit 2 are reserved. |
| Response type indicator | Byte 1 bit 3 | 0 means that this is a positive response.<br>1 means that this is a negative response. The sense data included indicator (SDI) must also be set if this bit is set. |
| (Reserved field) | Byte 1 bits 4-6 | Reserved. |

(continued on next page)

**Table C–3 (Cont.) Response Header**

| Field | Location | Values and Functions |
|---|---|---|
| Pacing indicator | Byte 1<br>bit 7 | 0 means that this is not a pacing response.<br>1 means that this is a pacing response. Indication of pacing response can be sent alone (isolated pacing response) or can be included in another response. |
| (Reserved field) | Byte 2 | Reserved. |

# D

# Definitions for the Application Programming Interface

Table D–1 presents symbols, values, and meanings to use when you write your application. Digital recommends that you use the definition files that accompany the API. 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.

**Table D–1 Definitions for the API**

| Symbol | Value | Meaning |
|---|---|---|
| SNAEVT$K_COMERR | 4 | Network communication error |
| SNAEVT$K_PLURESET | 5 | PLU reset session by sending CLEAR |
| SNAEVT$K_RCVEXP | 1 | Data received on expedited flow |
| SNAEVT$K_TERM | 3 | Link termination by IBM host or SNA Gateway |
| SNAEVT$K_UNBHLD | 2 | UNBIND type 2 sent by IBM host |
| SNALU0$K_ACTIVE | 1 | Active connect |
| SNALU0$K_EXPEDITED_FLOW | 1 | Expedited flow |
| SNALU0$K_MAX | 4 | Maximum message class code |
| SNALU0$K_MCLASS_DFC | 2 | Message class data flow control |
| SNALU0$K_MCLASS_FORMATTED_FM | 0 | Formatted user data |

(continued on next page)

**Table D–1 (Cont.) Definitions for the API**

| Symbol | Value | Meaning |
|---|---|---|
| SNALU0$K_MCLASS_NETWORK_CONTROL | 1 | Network control request unit |
| SNALU0$K_MCLASS_SESSION_CONTROL | 3 | Session control request unit |
| SNALU0$K_MCLASS_UNFORMATTED_FM | 4 | Unformatted user data |
| SNALU0$K_MIN | 0 | Minimum message class code |
| SNALU0$K_MTYPE_BID | 200 | Bid |
| SNALU0$K_MTYPE_BIS | 112 | Bracket initiation stopped |
| SNALU0$K_MTYPE_CANCEL | 131 | Cancel |
| SNALU0$K_MTYPE_CHASE | 132 | Chase |
| SNALU0$K_MTYPE_CLEAR | 161 | Clear |
| SNALU0$K_MTYPE_LUSTAT | 4 | LU status |
| SNALU0$K_MTYPE_QC | 129 | Quiesce complete |
| SNALU0$K_MTYPE_QEC | 128 | Quiesce at end of chain |
| SNALU0$K_MTYPE_RELQ | 130 | Release quiesce |
| SNALU0$K_MTYPE_RQR | 163 | Request recover |
| SNALU0$K_MTYPE_RSHUTD | 194 | Request shutdown |
| SNALU0$K_MTYPE_RTR | 5 | Ready to receive |
| SNALU0$K_MTYPE_SBI | 113 | Stop bracket initiation |
| SNALU0$K_MTYPE_SDT | 160 | Start data traffic |
| SNALU0$K_MTYPE_SHUTC | 193 | Shutdown complete |
| SNALU0$K_MTYPE_SHUTD | 192 | Shutdown |
| SNALU0$K_MTYPE_SIG | 201 | Signal |
| SNALU0$K_MTYPE_STSN | 162 | Set and test sequence number |
| SNALU0$K_NEGATIVE_RSP | 1 | Negative response |
| SNALU0$K_NORMAL_FLOW | 0 | Normal flow |
| SNALU0$K_PASSIVE | 0 | Passive connect |
| SNALU0$K_POSITIVE_RSP | 0 | Positive response |
| SNALU0$K_REQUEST | 0 | Request |

(continued on next page)

**Table D–1 (Cont.) Definitions for the API**

| Symbol | Value | Meaning |
|---|---|---|
| SNALU0$K_RESPONSE | 1 | Response |
| SNALU0$K_RSP_NONE | 0 | No response |
| SNALU0$K_RSP_RQD1 | 8 | Definite response required |
| SNALU0$K_RSP_RQD2 | 2 | Definite response required |
| SNALU0$K_RSP_RQD3 | 10 | Definite response required |
| SNALU0$K_RSP_RQE1 | 9 | Exception response |
| SNALU0$K_RSP_RQE2 | 3 | Exception response |
| SNALU0$K_RSP_RQE3 | 11 | Exception response |
| SNALU0$K_ST_BSM_BETB | 1 | Between brackets |
| SNALU0$K_ST_BSM_INB | 2 | In brackets |
| SNALU0$K_ST_BSM_P_BB | 3 | Between brackets pending |
| SNALU0$K_ST_BSM_P_INB | 4 | In brackets pending |
| SNALU0$K_ST_BSM_P_TERM_R | 6 | Pending bracket termination receive |
| SNALU0$K_ST_BSM_P_TERM_S | 5 | Pending bracket termination send |
| SNALU0$K_ST_CHAIN_BETC | 1 | Between chain |
| SNALU0$K_ST_CHAIN_INC | 2 | In chain |
| SNALU0$K_ST_CHAIN_PURGE | 3 | Purge chain |
| SNALU0$K_ST_QEC_PEND | 2 | Quiesce pending |
| SNALU0$K_ST_QEC_QUIESCED | 3 | Quiesced |
| SNALU0$K_ST_QEC_RESET | 1 | Quiesce reset |
| SNALU0$K_ST_SES_ACTIVE | 3 | Session active |
| SNALU0$K_ST_SES_P_ACTIVE | 2 | Active session pending |
| SNALU0$K_ST_SES_P_RESET | 4 | Session reset pending |
| SNALU0$K_ST_SES_RESET | 1 | Session reset |
| SNALU0$K_ST_TURN_CONT | 1 | Contention |
| SNALU0$K_ST_TURN_CONT_R | 3 | Contention receive |
| SNALU0$K_ST_TURN_CONT_S | 2 | Contention send |

(continued on next page)

**Table D–1 (Cont.) Definitions for the API**

| Symbol | Value | Meaning |
|---|---|---|
| SNALU0$K_ST_TURN_ERPR | 8 | Error recovery receive |
| SNALU0$K_ST_TURN_ERPS | 7 | Error recovery send |
| SNALU0$K_ST_TURN_RCV | 5 | Receive |
| SNALU0$K_ST_TURN_RCV_81B | 6 | Received 081B sense code |
| SNALU0$K_ST_TURN_SEND | 4 | Send |

# E
# Programming Examples

This appendix contains programming examples designed to show you how to make calls to the Digital SNA Application Programming Interface (API) for OpenVMS in your OpenVMS applications. One complete programming example and fragments of several programs illustrate how to use the API. In addition, the examples provide tips that will help you solve problems you may encounter in using the different languages. The examples use the following languages:

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

Numbered callouts refer to comments at the end of each programming example. Where the same numbered callout appears twice within an example, the comment applies to all the text between the two numbered callouts.

## E.1 FORTRAN Programming Example

This program connects to CICS and activates the CSFE transaction (a remote loopback program). It prompts you for the Gateway DECnet node name and the CICS access name. The application then establishes a connection with CICS and requests the CSFE transaction. After the CSFE instruction screen is received, some test data is generated. The data is converted to EBCDIC and sent to CSFE. The data is echoed from CSFE and the session is disconnected.

```
        PROGRAM LU0_EXAMPLE
C
C Declaration
C
    [1] INCLUDE 'SYS$LIBRARY:SNALU0DEF/NOLIST'
    [1] INCLUDE 'SYS$LIBRARY:SNALIBDEF/NOLIST'

    [2] EXTERNAL  NOTIFY_RTN           ! Define notify routine

        PARAMETER (BUFFER_LENGTH = 4103,
     1            LU0_EFN = 10)
        INTEGER*2 REQ_IND, MORE_DATA,MSG_CLASS, MSG_TYPE,FLOW,ALT_CODE
        INTEGER*2 BEG_BRACKET, END_BRACKET, SENSE_INC, RESP_TYPE
        INTEGER*2 END_DATA, FIRST_SEQ_NUM, LAST_SEQ_NUM
        INTEGER*4 SESSION_ID, DATA_LENGTH, SESSION_ADDRESS
        INTEGER*4 RETURN_CODE
        INTEGER*4 STATUS_VECTOR(SNALU0$K_MIN_STATUS_VECTOR)
        INTEGER*4 NOTIFY_VECTOR(SNALU0$K_MIN_NOTIFY_VECTOR)
        INTEGER*4 ISTAT
        CHARACTER*4 ASCII_TPN_NAME/'CSFE'/
        CHARACTER*8 NODE_NAME, ACC_NAME
        CHARACTER*52 CANNED_MESSAGE
     1    /'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'/
        CHARACTER*(BUFFER_LENGTH) DATA_BUFFER
        PARAMETER (SESSION_ADDRESS = 0)
C
C Global data
C
    [3] COMMON /NOTIFY/NOTIFY_VECTOR
```

```
C
C Get gateway node and access name
C
        TYPE 9001                       ! Prompt for gateway node
        ACCEPT 9002, NODE_NAME          ! Input gateway node
        TYPE 9003                       ! Prompt for access name
        ACCEPT 9002, ACC_NAME           ! Input access name

C
C Request connection
C
        RETURN_CODE = SNALU0$REQUEST_CONNECT_W(SESSION_ID,
     1    %DESCR(STATUS_VECTOR),
     2    %REF(SNALU0$K_ACTIVE),
     3    NODE_NAME,
     4    ACC_NAME,,
     5    SESSION_ADDRESS,,,,,, [4]
     6    NOTIFY_RTN,
     7    SESSION_ID,
     8    %DESCR(NOTIFY_VECTOR),
     9    DATA_BUFFER,
     A    DATA_LENGTH,
     B    %REF(LU0_EFN))

        IF (.NOT.RETURN_CODE) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP 'CONNECT failed'
        ENDIF

C*******************************************************************
C*                                                                 *
C* Acknowledge Bind - in this example we assume the BIND is        *
C* satisfactory. Normally you would have to examine the bind image  *
C* to verify that your application can handle the session defined by *
C* the BIND.                                                       *
C*                                                                 *
C* A negotiable BIND would be effected by modifying the BIND image  *
C* and positively responding.                                      *
C*                                                                 *
C*******************************************************************
        CALL XMIT_POS_RESP(SESSION_ID, STATUS_VECTOR, DATA_BUFFER,
     1    DATA_LENGTH)
```

```
C
C Receive start data traffic (SDT)
C
        RETURN_CODE = SNALU0$RECEIVE_MESSAGE_W(SESSION_ID,
     1    %DESCR(STATUS_VECTOR),
     2    DATA_BUFFER,
     3    DATA_LENGTH,
     4    REQ_IND,
     5    MORE_DATA,
     6    MSG_CLASS,
     7    MSG_TYPE,
     8    FLOW,
     9    ALT_CODE,
     A    BEG_BRACKET,
     B    END_BRACKET,
     C    SENSE_INC,
     D    RESP_TYPE,
     E    END_DATA,
     F    FIRST_SEQ_NUM,
     G    LAST_SEQ_NUM,
     H    %REF(LU0_EFN))

        IF ((.NOT.RETURN_CODE).OR.(MSG_TYPE.NE.SNALU0$K_MTYPE_SDT))
     1    THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP 'Failed to receive SDT'
        ENDIF

C
C Acknowledge SDT
C
        CALL XMIT_POS_RESP(SESSION_ID, STATUS_VECTOR, DATA_BUFFER,
     1    DATA_LENGTH)
```

```
C
C Receive BID
C
        RETURN_CODE = SNALU0$RECEIVE_MESSAGE_W(SESSION_ID,
     1    %DESCR(STATUS_VECTOR),
     2    DATA_BUFFER,
     3    DATA_LENGTH,
     4    REQ_IND,
     5    MORE_DATA,
     6    MSG_CLASS,
     7    MSG_TYPE,
     8    FLOW,
     9    ALT_CODE,
     A    BEG_BRACKET,
     B    END_BRACKET,
     C    SENSE_INC,
     D    RESP_TYPE,
     E    END_DATA,
     F    FIRST_SEQ_NUM,
     G    LAST_SEQ_NUM,
     H    %REF(LU0_EFN))

        IF (.NOT.RETURN_CODE) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP 'Failed to receive initial bid'
        ENDIF

C
C Acknowledge BID
C
        CALL XMIT_POS_RESP(SESSION_ID, STATUS_VECTOR, DATA_BUFFER,
     1    DATA_LENGTH)

C
C Receive CICS logo
C
        RETURN_CODE = SNALU0$RECEIVE_MESSAGE_W(SESSION_ID,
     1    %DESCR(STATUS_VECTOR),
     2    DATA_BUFFER,
     3    DATA_LENGTH,
     4    REQ_IND,
     5    MORE_DATA,
     6    MSG_CLASS,
     7    MSG_TYPE,
     8    FLOW,
     9    ALT_CODE,
     A    BEG_BRACKET,
     B    END_BRACKET,
     C    SENSE_INC,
     D    RESP_TYPE,
     E    END_DATA,
     F    FIRST_SEQ_NUM,
     G    LAST_SEQ_NUM,
```

```
     H    %REF(LU0_EFN))

        IF (.NOT.RETURN_CODE) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP 'Failed to receive CICS logo'
        ENDIF

C
C Acknowledge CICS logo
C
        CALL XMIT_POS_RESP(SESSION_ID, STATUS_VECTOR, DATA_BUFFER,
     1    DATA_LENGTH)

C
C For this test program, the EBI or CDI should be received on this
C call. If an EBI or CDI is not received, then there is more data to
C be received. Usually, the data that has already been received is
C used to receive the rest of the data.
C
        IF ((END_BRACKET .NE. 1) .AND. (END_DATA .NE. 1)) THEN
            RETURN_CODE = SNALU0$RECEIVE_MESSAGE_W(SESSION_ID,
     1        %DESCR(STATUS_VECTOR),
     2        DATA_BUFFER,
     3        DATA_LENGTH,
     4        REQ_IND,
     5        MORE_DATA,
     6        MSG_CLASS,
     7        MSG_TYPE,
     8        FLOW,
     9        ALT_CODE,
     A        BEG_BRACKET,
     B        END_BRACKET,
     C        SENSE_INC,
     D        RESP_TYPE,
     E        END_DATA,
     F        FIRST_SEQ_NUM,
     G        LAST_SEQ_NUM,
     H        %REF(LU0_EFN))

            IF (.NOT.RETURN_CODE) THEN
                    ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                    STOP 'Failed to receive CICS logo'
            ENDIF
```

```
C
C Acknowledge rest of CICS logo
C
            CALL XMIT_POS_RESP(SESSION_ID, STATUS_VECTOR, DATA_BUFFER,
     1        DATA_LENGTH)
        ENDIF

C
C Send "clear screen" request
C
        DATA_LENGTH = SNABUF$K_HDLEN+1
        DATA_BUFFER(8:8) = CHAR('6D'X)          ! EBCDIC code for clear
    [5] MSG_CLASS = SNALU0$K_MCLASS_UNFORMATTED_FM

        RETURN_CODE = SNALU0$TRANSMIT_MESSAGE_W(SESSION_ID,
     1    %DESCR(STATUS_VECTOR),
     2    DATA_BUFFER,
     3    DATA_LENGTH,
     4    MSG_CLASS,
     5    %REF(.FALSE.),
     6    %REF(.FALSE.),
     7    %REF(SNALU0$K_RSP_RQE1),
     8    %REF(.FALSE.),
     9    %REF(.FALSE.),
     A    FIRST_SEQ_NUM,
     B    LAST_SEQ_NUM,
     C    %REF(LU0_EFN))

        IF (.NOT.RETURN_CODE) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP 'TRANSMIT_MESSAGE failed'
        ENDIF
```

```
C
C Receive clear screen command
C
        RETURN_CODE = SNALU0$RECEIVE_MESSAGE_W(SESSION_ID,
     1    %DESCR(STATUS_VECTOR),
     2    DATA_BUFFER,
     3    DATA_LENGTH,
     4    REQ_IND,
     5    MORE_DATA,
     6    MSG_CLASS,
     7    MSG_TYPE,
     8    FLOW,
     9    ALT_CODE,
     A    BEG_BRACKET,
     B    END_BRACKET,
     C    SENSE_INC,
     D    RESP_TYPE,
     E    END_DATA,
     F    FIRST_SEQ_NUM,
     G    LAST_SEQ_NUM,
     H    %REF(LU0_EFN))

        IF (.NOT.RETURN_CODE) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP 'Failed to receive clear screen command'
        ENDIF

C
C Acknowledge clear screen command
C
        CALL XMIT_POS_RESP(SESSION_ID, STATUS_VECTOR, DATA_BUFFER,
     1    DATA_LENGTH)

C
C Transmit transaction name (CSFE)
C
        CALL SEND_MESSAGE(SESSION_ID, STATUS_VECTOR, ASCII_TPN_NAME,
     1    DATA_BUFFER)
```

```
C
C Receive CSFE operator screen
C
        RETURN_CODE = SNALU0$RECEIVE_MESSAGE_W(SESSION_ID,
     1    %DESCR(STATUS_VECTOR),
     2    DATA_BUFFER,
     3    DATA_LENGTH,
     4    REQ_IND,
     5    MORE_DATA,
     6    MSG_CLASS,
     7    MSG_TYPE,
     8    FLOW,
     9    ALT_CODE,
     A    BEG_BRACKET,
     B    END_BRACKET,
     C    SENSE_INC,
     D    RESP_TYPE,
     E    END_DATA,
     F    FIRST_SEQ_NUM,
     G    LAST_SEQ_NUM,
     H    %REF(LU0_EFN))

        IF (.NOT.RETURN_CODE) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP 'RECEIVE_MESSAGE failed'
        ENDIF

C
C Acknowledge CSFE operator screen
C
        CALL XMIT_POS_RESP(SESSION_ID, STATUS_VECTOR, DATA_BUFFER,
     1    DATA_LENGTH)

C
C Send test message
C
        CALL SEND_MESSAGE(SESSION_ID, STATUS_VECTOR, CANNED_MESSAGE,
     1    DATA_BUFFER)
```

```
C
C Receive test message
C
        RETURN_CODE = SNALU0$RECEIVE_MESSAGE_W(SESSION_ID,
     1    %DESCR(STATUS_VECTOR),
     2    DATA_BUFFER,
     3    DATA_LENGTH,
     4    REQ_IND,
     5    MORE_DATA,
     6    MSG_CLASS,
     7    MSG_TYPE,
     8    FLOW,
     9    ALT_CODE,
     A    BEG_BRACKET,
     B    END_BRACKET,
     C    SENSE_INC,
     D    RESP_TYPE,
     E    END_DATA,
     F    FIRST_SEQ_NUM,
     G    LAST_SEQ_NUM,
     H    %REF(LU0_EFN))

        IF (.NOT.RETURN_CODE) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP 'RECEIVE_MESSAGE failed'
        ENDIF

C
C Acknowledge data  [6]
C
        CALL XMIT_POS_RESP(SESSION_ID, STATUS_VECTOR, DATA_BUFFER,
     1    DATA_LENGTH)

C
C Disconnect session  [7]
C
        RETURN_CODE = SNALU0$REQUEST_DISCONNECT_W(SESSION_ID,
     1    %DESCR(STATUS_VECTOR))
        IF (.NOT.RETURN_CODE) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP 'DISCONNECT failed'
        ENDIF

        ISTAT = SYS$PUTMSG(STATUS_VECTOR)
        STOP
```

```
C
C Format statements
C
9001    FORMAT(1X,'Enter gateway node: ',2X,$)
9002    FORMAT(A8)
9003    FORMAT(1X,'Enter access name: ',2X,$)
        END

        SUBROUTINE XMIT_POS_RESP(SESSION_ID,STATUS_VECTOR,BUFFER,LENGTH)

        INCLUDE 'SYS$LIBRARY:SNALU0DEF/NOLIST'
        INTEGER*4 STATUS_VECTOR(SNALU0$K_MIN_STATUS_VECTOR)
        INTEGER*4 SESSION_ID, LENGTH, RETURN_CODE
        INTEGER*4 ISTAT
        CHARACTER*(*) BUFFER

        RETURN_CODE = SNALU0$TRANSMIT_RESPONSE_W(SESSION_ID,
     1    %DESCR(STATUS_VECTOR),
     2    BUFFER,
     3    LENGTH,
     4    %REF(SNALU0$K_POSITIVE_RSP))

        IF (.NOT.RETURN_CODE) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP 'TRANSMIT_RESPONSE failed'
        ENDIF

        RETURN
        END

        SUBROUTINE SEND_MESSAGE(SESSION_ID, STATUS_BLOCK, MESSAGE, BUFFER)

        INCLUDE 'SYS$LIBRARY:SNALU0DEF/NOLIST'
        INCLUDE 'SYS$LIBRARY:SNALIBDEF/NOLIST'

        INTEGER*4 STATUS_VECTOR(SNALU0$K_MIN_STATUS_VECTOR)
        INTEGER*4 SESSION_ID, RETURN_CODE
        INTEGER*4 FIRST_CHAR, LAST_CHAR, LENGTH, ISTAT
        INTEGER*2 MSG_CLASS/SNALU0$K_MCLASS_UNFORMATTED_FM/
        CHARACTER*(*) MESSAGE, BUFFER

C
C Translate data to EBCDIC
C
    [8] FIRST_CHAR = SNABUF$K_HDLEN + 3 + 1     [9]
        LAST_CHAR = FIRST_CHAR + LEN(MESSAGE) - 1
        LENGTH = LAST_CHAR
   [10] ISTAT = LIB$TRA_ASC_EBC(MESSAGE, BUFFER(FIRST_CHAR:LAST_CHAR))

        IF (.NOT.ISTAT) STOP 'LIB$TRA_ASC_EBC failed'

        BUFFER(8:8) = CHAR('7D'X)               ! Enter key code
        BUFFER(9:9) = CHAR('40'X)               ! Cursor Position
        BUFFER(10:10) = CHAR('C4'X)             ! = CHAR(04) (encoded)
```

```
C
C Transmit test data
C
        RETURN_CODE = SNALU0$TRANSMIT_MESSAGE_W(SESSION_ID,
     1    %DESCR(STATUS_VECTOR),
     2    BUFFER,
     3    LENGTH,
     4    MSG_CLASS,
     5    %REF(.FALSE.),
     6    %REF(.FALSE.),
     7    %REF(SNALU0$K_RSP_RQE1),              ! resp_type
     8    %REF(.FALSE.),
     9    %REF(.FALSE.),
     A    FIRST_SEQ_NUM,
     B    LAST_SEQ_NUM,
     C    %REF(LU0_EFN))

        IF (.NOT.RETURN_CODE) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP 'TRANSMIT_MESSAGE failed'
        ENDIF

        RETURN
        END

        SUBROUTINE NOTIFY_RTN(EVENT_CODE, NOTIFY_PARM)

        INCLUDE 'SYS$LIBRARY:SNALU0DEF/NOLIST'
        INCLUDE 'SYS$LIBRARY:SNALIBDEF/NOLIST'

        INTEGER*4 NOTIFY_VECTOR(SNALU0$K_MIN_NOTIFY_VECTOR)
        INTEGER*4 EVENT_CODE
        CHARACTER*31 RCVEXP_EVENT/'Data received on expedited flow'/
        CHARACTER*27 COMERR_EVENT/'Gateway communication error'/
        CHARACTER*30 TERM_EVENT/'Session terminated by IBM host'/
        CHARACTER*33 PLURESET_EVENT/'Half session state machines reset'/
        CHARACTER*22 UNBHLD_EVENT/'Unbind type 2 received'/
        CHARACTER*35 UNKNOWN_EVENT/'Unknown asynchronous event reported'/

C
C Global data
C
        COMMON /NOTIFY/NOTIFY_VECTOR
```

```
      [11] IF (EVENT_CODE .EQ. SNAEVT$K_RCVEXP) THEN
              TYPE 9100, RCVEXP_EVENT
           ELSEIF (EVENT_CODE .EQ. SNAEVT$K_UNBHLD) THEN
              TYPE 9100, UNBHLD_EVENT
           ELSEIF (EVENT_CODE .EQ. SNAEVT$K_TERM) THEN
              TYPE 9100, TERM_EVENT
           ELSEIF (EVENT_CODE .EQ. SNAEVT$K_COMERR) THEN
              TYPE 9100, COMERR_EVENT
           ELSEIF (EVENT_CODE .EQ. SNAEVT$K_PLURESET) THEN
              TYPE 9100, PLURESET_EVENT
           ELSE
              TYPE 9100, UNKNOWN_EVENT
           ENDIF
           ISTAT = SYS$PUTMSG(NOTIFY_VECTOR)
           RETURN
9100 [11] FORMAT(3X,'Asynchronous notification: ',A35//)
           END
```

**COMMENTS**

1. Include the LU0 and basic API symbol definition libraries.
2. Define the notify routine externally so the compiler knows it is an address and not a variable.
3. Note that the notify vector is global. This enables the notify routine and main program to access it. The vector is not passed to the notify routine.
4. Commas indicate that you do not want to specify values for the parameters and will accept the default values provided by the Interface.
5. Be sure to set the message class to unformatted otherwise the format indicator will be set, and IBM will reject the message.
6. A more complete program would verify the data stream before acknowledging it.
7. This is an abrupt session termination that may cause error logging on some IBM subsystems. See other examples for a more orderly session termination.
8. The actual data (CSFE) starts in position 11 of the buffer. The first 7 bytes are reserved for the API header and the next 3 bytes contain data stream control characters (aid key and cursor position).

9. SNABUF$K_HDLEN is a literal defining the API header length in the SNALIBDEF or SNALU0DEF file. The extra 4 bytes position the message text after the header and 3 bytes of data stream control characters.
10. You can use OpenVMS Library routines to do parts of your application, such as translating ASCII to EBCDIC or vice versa.
11. Literals for event codes are defined in SNALIBDEF or SNALU0DEF.

## E.2 FORTRAN Definition Files

The FORTRAN definition files, SYS$LIBRARY:SNALU0DEF.FOR and SYS$LIBRARY:SNALIBDEF.FOR, were built for FORTRAN compilers supporting structure definitions. If you plan to use a FORTRAN compiler which does not have this support, you can edit the definition files to remove the structure definitions. You should comment out or delete all lines between the STRUCTURE and END STRUCTURE keywords except for PARAMETER statements.

## E.3 COBOL Programming Example

This program connects to CICS and activates the CSFE transaction (a remote loopback program). It prompts you for the Gateway DECnet node or TCP/IP host name and the CICS access name. The application then establishes a connection with CICS and requests the CSFE transaction. After the CSFE instruction screen is received, some test data is generated. The data is converted to EBCDIC and sent to CSFE. The data is echoed from CSFE and the session is disconnected.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LU0TEST.

DATA DIVISION.

*
* Declaration
*
WORKING-STORAGE SECTION.
[1] ACCNAM              PIC X(08)  VALUE SPACES.
01  ALT-CODE            PIC 9(04)  COMP.
01  BEG-BRACKET         PIC 9(04)  COMP.
01  BIND-BUF-LEN        PIC 9(08)  COMP.
01  END-BRACKET         PIC S9(04) COMP.
01  END-DATA            PIC S9(04) COMP.
01  FLOW                PIC 9(04)  COMP.
01  IDX1                PIC 9(04)  COMP.
01  IDX2                PIC 9(04)  COMP.
01  MORE-DATA           PIC 9(04)  COMP.
01  MSG-CLASS           PIC 9(04)  COMP.
01  MSG-TYPE            PIC 9(04)  COMP.
01  NODNAM              PIC X(06)  VALUE SPACES.
01  NOTIFY-RTN-NAME     PIC X(06)  VALUE "NOTIFY".
01  NOTIFY-RTN-ADDR     PIC 9(09)  COMP.
01  NOTIFY-VEC          PIC X(64).
01  NUM-BYTES-CONVERT   PIC 9(04)  COMP.
01  PLU-FIRST-SEQ-NUM   PIC 9(04)  COMP.
01  PLU-LAST-SEQ-NUM    PIC 9(04)  COMP.
01  REQ-IND             PIC 9(04)  COMP.
01  RESP-TYPE           PIC 9(04)  COMP.
01  SENSE-INC           PIC 9(04)  COMP.
01  SESS-ID             PIC 9(08)  COMP.
01  SESSION-ADDRESS     PIC 9(08)  COMP VALUE 0.
01  SLU-FIRST-SEQ-NUM   PIC 9(04)  COMP.
01  SLU-LAST-SEQ-NUM    PIC 9(04)  COMP.
01  SS-STATUS           PIC S9(09) COMP.
01  STATUS-VEC          PIC X(64).
[1] 01  TURN-RETAIN         PIC 9(04)  COMP.
01  BIND-BUFFER.
```

```
    02  BIND-BUF1       PIC 9(07).
    02  BIND-BUF2.
        03 BIND-BUFFER2 OCCURS 4096 TIMES PIC X.
01  DATA-BUFFER.
    02  DATA-BUF        OCCURS 10 TIMES PIC X(02).
01  TEMP-LONGWORD.
    02  TEMP-BYTES      OCCURS 4 TIMES PIC 9(01).

01  TEST-DATA.
    02  TST-DATA        OCCURS 4103 TIMES PIC X.

*
*  SNA symbol definitions
*
01 SNALU0$K_ACTIVE      PIC 9(08)  COMP VALUE 1.
01 SNALU0$K_HDLEN       PIC 9(08)  COMP VALUE 7.
01 SNALU0$K_MCLASS_UNFORMATTED_FM
                        PIC 9(08)  COMP VALUE 4.
01 SNALU0$K_MTYPE_BID   PIC 9(08)  COMP VALUE 200.
01 SNALU0$K_MTYPE_SDT   PIC 9(08)  COMP VALUE 160.
01 SNALU0$K_POSITIVE_RSP
                        PIC 9(08)  COMP VALUE 0.
01 SNALU0$K_RSP_RQE1    PIC 9(08)  COMP VALUE 9.

PROCEDURE DIVISION.

*
* Main program
*
MAIN.
[2] PERFORM GET-NODE-ACC-NAME.
    PERFORM GET-NOTIFY-RTN-ADDR.
    PERFORM REQUEST-CONNECT.
```

```
**********************************************************************
*                                                                    *
* Acknowledge Bind - in this example we assume the BIND is           *
* satisfactory. Normally you would have to examine the bind image to *
* verify that your application can handle the session defined by the *
* BIND.                                                              *
*                                                                    *
* A negotiable BIND would be effected by modifying the BIND image and *
* positively responding.                                             *
*                                                                    *
**********************************************************************
    PERFORM TRANSMIT-RESPONSE.
    PERFORM RECEIVE-MESSAGE.
    IF (MSG-TYPE IS EQUAL TO SNALU0$K_MTYPE_SDT)
      THEN
        PERFORM TRANSMIT-RESPONSE
      ELSE
        PERFORM EXIT-PROGRAM.
[2] PERFORM RECEIVE-MESSAGE.
    IF (MSG-TYPE IS EQUAL TO SNALU0$K_MTYPE_BID)
      THEN
        PERFORM TRANSMIT-RESPONSE
      ELSE
        PERFORM EXIT-PROGRAM.
    PERFORM RECEIVE-MESSAGE.
    PERFORM TRANSMIT-RESPONSE.

**********************************************************************
* For this test program, the EBI or CDI should be received on this   *
* call. If an EBI or CDI is not received, then there is more data to *
* be received. Usually, the data that has already been received is  *
* used to receive the rest of the data.                              *
**********************************************************************
    IF (END-BRACKET IS FAILURE) AND (END-DATA IS FAILURE)
    THEN
        PERFORM RECEIVE-MESSAGE
        PERFORM TRANSMIT-RESPONSE.
    PERFORM CLEAR-SCREEN.
    PERFORM TRANSMIT-MESSAGE.
    PERFORM RECEIVE-MESSAGE.
    PERFORM TRANSMIT-RESPONSE.
    PERFORM CSFE.
    PERFORM TRANSMIT-MESSAGE.
    PERFORM RECEIVE-MESSAGE.
    PERFORM TRANSMIT-RESPONSE.
    PERFORM CONVERT-DATA.
    PERFORM TRANSMIT-MESSAGE.
    PERFORM RECEIVE-MESSAGE.
    PERFORM TRANSMIT-RESPONSE.
    PERFORM REQUEST-DISCONNECT.
    PERFORM EXIT-PROGRAM.
```

```
*
* Get node name and access name
*
GET-NODE-ACC-NAME.
    DISPLAY "Enter gateway node: " WITH NO ADVANCING.
    ACCEPT NODNAM.
    DISPLAY "Enter access name: " WITH NO ADVANCING.
    ACCEPT ACCNAM.

*
* Get address of notify routine
*
GET-NOTIFY-RTN-ADDR.
    CALL "COB$CALL" USING BY DESCRIPTOR NOTIFY-RTN-NAME
                    GIVING NOTIFY-RTN-ADDR.

*
* Request a connection with the IBM
*
REQUEST-CONNECT.
    CALL "SNALU0$REQUEST_CONNECT_W" USING
                                BY REFERENCE SESS-ID,
                                BY DESCRIPTOR STATUS-VEC,
                                BY REFERENCE SNALU0$K_ACTIVE,
                                BY DESCRIPTOR NODNAM, ACCNAM,
                                BY VALUE 0,
                                BY REFERENCE SESSION-ADDRESS,
                                BY VALUE 0,0,0,0,0,
                                BY VALUE NOTIFY-RTN-ADDR,
                                BY VALUE 0,
                                BY DESCRIPTOR NOTIFY-VEC,
                                BY DESCRIPTOR BIND-BUFFER,
                                BY REFERENCE BIND-BUF-LEN,
                                BY VALUE 0,0,0,
                                GIVING SS-STATUS.

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

*
* Respond to the last RU received
*
TRANSMIT-RESPONSE.
    CALL "SNALU0$TRANSMIT_RESPONSE_W" USING
                                BY REFERENCE SESS-ID,
                                BY DESCRIPTOR STATUS-VEC,
                                BY DESCRIPTOR BIND-BUFFER,
                                BY REFERENCE BIND-BUF-LEN,
                                BY REFERENCE SNALU0$K_POSITIVE_RSP,
                                BY VALUE 0,0,0,0
                                GIVING SS-STATUS.
```

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

*
* Receive a message from the IBM
*
RECEIVE-MESSAGE.
    CALL "SNALU0$RECEIVE_MESSAGE_W" USING
                                BY REFERENCE SESS-ID,
                                BY DESCRIPTOR STATUS-VEC,
                                BY DESCRIPTOR BIND-BUFFER,
                                BY REFERENCE BIND-BUF-LEN,
                                BY REFERENCE REQ-IND,
                                BY REFERENCE MORE-DATA,
                                BY REFERENCE MSG-CLASS,
                                BY REFERENCE MSG-TYPE,
                                BY REFERENCE FLOW,
                                BY REFERENCE ALT-CODE,
                                BY REFERENCE BEG-BRACKET,
                                BY REFERENCE END-BRACKET,
                                BY REFERENCE SENSE-INC,
                                BY REFERENCE RESP-TYPE,
                                BY REFERENCE END-DATA,
                                BY REFERENCE PLU-FIRST-SEQ-NUM,
                                BY REFERENCE PLU-LAST-SEQ-NUM,
                                BY VALUE 0,0,0
                                GIVING SS-STATUS.

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

```
*
* Transmit "clear screen"
*
CLEAR-SCREEN.
    MOVE 8    TO BIND-BUF-LEN.
[3] MOVE "6D" TO DATA-BUFFER.
    MOVE 0    TO ALT-CODE.
    MOVE 0    TO END-BRACKET.
    MOVE 0    TO MORE-DATA.
    MOVE 0    TO TURN-RETAIN.
    MOVE 2    TO NUM-BYTES-CONVERT.
    PERFORM CONVERT-TO-HEX.

*
* Transmit message to IBM
*
TRANSMIT-MESSAGE.
    CALL "SNALU0$TRANSMIT_MESSAGE_W" USING
                        BY REFERENCE SESS-ID,
                        BY DESCRIPTOR STATUS-VEC,
                        BY DESCRIPTOR BIND-BUFFER,
                        BY REFERENCE BIND-BUF-LEN,
                        BY REFERENCE SNALU0$K_MCLASS_UNFORMATTED_FM,
                        BY REFERENCE ALT-CODE,
                        BY REFERENCE END-BRACKET,
                        BY REFERENCE SNALU0$K_RSP_RQE1,
                        BY REFERENCE MORE-DATA,
                        BY REFERENCE TURN-RETAIN,
                        BY REFERENCE SLU-FIRST-SEQ-NUM,
                        BY REFERENCE SLU-LAST-SEQ-NUM,
                        BY VALUE 0,0,0
                        GIVING SS-STATUS.

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

```
*
* Convert "CSFE" to hex.
*
CSFE.
    MOVE 0                TO ALT-CODE.
    MOVE 0                TO END-BRACKET.
    MOVE 0                TO MORE-DATA.
    MOVE 0                TO TURN-RETAIN.
    MOVE "          CSFE" TO TEST-DATA.
    PERFORM TRANSLATE-ASC-EBC.
[4] MOVE "7D40C4"         TO DATA-BUFFER.
    MOVE 6                TO NUM-BYTES-CONVERT.
    PERFORM CONVERT-TO-HEX.
    MOVE 14               TO BIND-BUF-LEN.

*
* Convert data to ebcdic.
*
CONVERT-DATA.
    MOVE 0        TO ALT-CODE.
    MOVE 0        TO END-BRACKET.
    MOVE 0        TO MORE-DATA.
    MOVE 0        TO TURN-RETAIN.
    MOVE "          ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
                  TO TEST-DATA.
    PERFORM TRANSLATE-ASC-EBC.
    MOVE "7D40C4" TO DATA-BUFFER.
    MOVE 6        TO NUM-BYTES-CONVERT.
    PERFORM CONVERT-TO-HEX.
    MOVE 62 TO BIND-BUF-LEN.

*
* Disconnect link
*
REQUEST-DISCONNECT.
    CALL "SNALU0$REQUEST_DISCONNECT_W" USING
                                BY REFERENCE SESS-ID,
                                BY DESCRIPTOR STATUS-VEC,
                                BY VALUE 0,0,0,0,0
                                GIVING SS-STATUS.

*
* Convert the specified byte from a text hex number to a hex value
*
CONVERT-TO-HEX.
    CALL "LIB$CVT_HTB" USING    BY VALUE NUM-BYTES-CONVERT,
                                BY REFERENCE DATA-BUFFER,
                                BY REFERENCE TEMP-LONGWORD,
                                GIVING SS-STATUS.
```

```
    IF SS-STATUS IS FAILURE
      THEN
        PERFORM  EXIT-PROGRAM.
    DIVIDE NUM-BYTES-CONVERT BY 2 GIVING NUM-BYTES-CONVERT.
    MOVE 1 TO IDX1.
    MOVE NUM-BYTES-CONVERT TO IDX2.
    PERFORM PLACE-IN-BIND-BUFFER NUM-BYTES-CONVERT TIMES.

*
* Store a byte into a specified location
*
PLACE-IN-BIND-BUFFER.
    MOVE TEMP-BYTES(IDX2) TO BIND-BUFFER2(IDX1).
    ADD 1 TO IDX1.
    SUBTRACT 1 FROM IDX2.

*
* Convert the specifed buffer from ASCII form to EBCDIC form
*
[5] TRANSLATE-ASC-EBC.
    CALL "LIB$TRA_ASC_EBC" USING
                                BY DESCRIPTOR TEST-DATA,
                                BY DESCRIPTOR BIND-BUFFER,
                                GIVING SS-STATUS.

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

*
* Exit the program with the status of the last call
*
EXIT-PROGRAM.
    CALL "SYS$PUTMSG" USING STATUS-VEC.
    STOP RUN.

IDENTIFICATION DIVISION.
PROGRAM-ID. NOTIFY INITIAL.

DATA DIVISION.

*
* Declaration
*
WORKING-STORAGE SECTION.

01 SNAEVT$K_RCVEXP      PIC 9(08)  COMP VALUE 1.
01 SNAEVT$K_UNBHLD      PIC 9(08)  COMP VALUE 2.
01 SNAEVT$K_TERM        PIC 9(08)  COMP VALUE 3.
01 SNAEVT$K_COMERR      PIC 9(08)  COMP VALUE 4.
01 SNAEVT$K_PLURESET    PIC 9(08)  COMP VALUE 5.

LINKAGE SECTION.
01 EVENT-CODE           PIC 9(08).
```

```
PROCEDURE DIVISION USING EVENT-CODE.
*
* Notify routine
*
BEGIN.
    IF EVENT-CODE IS EQUAL TO SNAEVT$K_RCVEXP
      THEN
        DISPLAY "Asynchronous notification: Data received on expedited flow".
    IF EVENT-CODE IS EQUAL TO SNAEVT$K_UNBHLD
      THEN
        DISPLAY "Asynchronous notification: Unbind type 2 received".
    IF EVENT-CODE IS EQUAL TO SNAEVT$K_TERM
      THEN
        DISPLAY "Asynchronous notification: Session terminated by IBM host".
    IF EVENT-CODE IS EQUAL TO SNAEVT$K_COMERR
      THEN
        DISPLAY "Asynchronous notification: Gateway communication error".
    IF EVENT-CODE IS EQUAL TO SNAEVT$K_PLURESET
      THEN
        DISPLAY "Asynchronous notification: Half session state machines reset".
    EXIT PROGRAM.
END PROGRAM NOTIFY.
END PROGRAM LU0TEST.
```

**COMMENTS**

1. Define the symbols you will need to write your application.
2. Break the application into simple procedures. Note that all of the procedures listed here are not shown in this programming fragment, but they are similar to those presented in this example.
3. 6D = an aid key (clear)
4. This data represents several pieces of information:
   - 7D = an aid key (enter)
   - 40C4 = the cursor address
5. You can use OpenVMS Library routines to do parts of your application, such as translating ASCII to EBCDIC and vice versa.

## E.4 MACRO Programming Example

This program connects to CICS and activates the CSFE transaction (a remote loopback program). It prompts you for the Gateway DECnet node or TCP/IP host name and the CICS access name. The application then establishes a connection with CICS and requests the CSFE transaction. After the CSFE instruction screen is received, some test data is generated. The data is converted to EBCDIC and sent to CSFE. The data is echoed from CSFE and the session is disconnected.

```
          .TITLE LU0MAR
       [1] LU0LIBDEF
       [1] SNALIBDF

.PSECT RWDATA,WRT,NOEXE,QUAD

;
; Declaration
;
TEST_SRC: .ASCID
          /          ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz/
ND_PROMP: .ASCID  /Enter gateway node: /
AC_PROMP: .ASCID  /Enter access name: /
STATUS :  .BLKL   10
SESS_ID:  .LONG   0                            ;session-id
STS_VEC:  .BLKB   SNALU0$K_MIN_STATUS_VECTOR    ;status-vector
STS_DSC:  .LONG   SNALU0$K_MIN_STATUS_VECTOR
          .ADDRESS STS_VEC
BIND_BUF: .BLKB   ^X1007                       ;bind-buffer
BIND_DSC: .LONG   ^X010E1007
          .ADDRESS BIND_BUF
BIND_LEN: .LONG   0                            ;bind-buffer-length
TEST_BUF: .BLKB   ^X003E
TEST_DSC: .LONG   ^X010E003E
          .ADDRESS TEST_BUF
ND_NAME:  .LONG   ^X020E0000                   ;node-name
          .ADDRESS 0
AC_NAME:  .LONG   ^X020E0000                   ;access-name
          .ADDRESS 0
REQ_IND:  .LONG   0
MORE_DATA:.LONG   0
MSG_CLASS:.LONG   0
MSG_TYP:  .LONG   0
FLOW:     .LONG   0
ALT_CODE: .LONG   0
BEG_BRAC: .LONG   0
END_BRAC: .LONG   0
SENS_INC: .LONG   0
SES_ADDR: .LONG   0
```

```
RESP_TYP: .LONG   0
END_DATA: .LONG   0
PLU_F_NO: .LONG   0
PLU_L_NO: .LONG   0
SLU_F_NO: .LONG   0
SLU_L_NO: .LONG   0
NT_VEC :  .BLKB   SNALU0$K_MIN_NOTIFY_VECTOR    ;notify-vector
NT_DSC:   .LONG   SNALU0$K_MIN_NOTIFY_VECTOR
          .ADDRESS NT_VEC
RCVEXP:   .ASCID
          /Asynchronous notification: Data received on expedited flow/
COMERR:   .ASCID
          /Asynchronous notification: Gateway communication error/
TERM:     .ASCID
          /Asynchronous notification: Session termination by IBM host/
PLURST:   .ASCID
          /Asynchronous notification: Half session state machines reset/
UNBHLD:   .ASCID
          /Asynchronous notification: Unbind type 2 received/
UNKNOWN:  .ASCID
          /Asynchronous notification: Unknown asynchronous event reported/

.PSECT CODE,NOWRT,EXE,LONG

;
; Notify routine
;
     [2] .ENTRY  NOTIFY$RTN, ^M<>    ;notify-routine entry point
         MOVL    4(AP), R1

         MOVAL   SNAEVT$K_RCVEXP, R2
         CMPL    R1, R2
         BNEQ    10$
         PUSHAQ  RCVEXP                       ;expedited-flow-received
         BRW     60$

10$:
         MOVAL   SNAEVT$K_UNBHLD, R2
         CMPL    R1, R2
         BNEQ    20$
         PUSHAQ  UNBHLD                       ;unbind type 2
         BRW     60$

20$:
         MOVAL   SNAEVT$K_TERM, R2
         CMPL    R1, R2
         BNEQ    30$
         PUSHAQ  TERM                         ;session termination
         BRW     60$
```

```
30$:
        MOVAL   SNAEVT$K_COMERR, R2
        CMPL    R1, R2
        BNEQ    40$
        PUSHAQ  COMERR                          ;communication error
        BRW     60$

40$:
        MOVAL   SNAEVT$K_PLURESET, R2
        CMPL    R1, R2
        BNEQ    50$
        PUSHAQ  PLURST                          ;plu-reset
        BRW     60$

50$:
        PUSHAQ  UNKNOWN                         ;unknown event

60$:
        CALLS   #1,G^LIB$PUT_OUTPUT             ;display the event
        $PUTMSG_S NT_VEC                        ;display notify-vector
        RET

;
;Main Program entry point
;
        .ENTRY  LU0MAR, ^M<>

;
; Get node name and access name
;
        PUSHAQ  ND_PROMP
        PUSHAQ  ND_NAME
        CALLS   #2,G^LIB$GET_INPUT
        BLBS    R0, 110$
        BRW     EXITS
```

```
110$:
        PUSHAQ  AC_PROMP
        PUSHAQ  AC_NAME
        CALLS   #2,G^LIB$GET_INPUT
        BLBS    R0, 120$
        BRW     EXITS

120$:
    [3] CLRL    R0
        PUSHL   #0                      ;ast parameter
        PUSHL   #0                      ;ast address
        PUSHL   #0                      ;event-flag
        PUSHAL  BIND_LEN                ;bind-buffer-length
        PUSHAQ  BIND_DSC                ;bind-buffer
        PUSHAQ  NT_DSC                  ;notify-vector
        PUSHL   #0                      ;notify-parameter
        PUSHAL  NOTIFY$RTN              ;notify-routine address
        PUSHL   #0                      ;data
        PUSHL   #0                      ;password
        PUSHL   #0                      ;user id
        PUSHL   #0                      ;logon mode
        PUSHL   #0                      ;application
        PUSHAL  SES_ADDR                ;session address
        PUSHL   #0                      ;circuit id
        PUSHAL  AC_NAME                 ;access-name
        PUSHAL  ND_NAME                 ;node-name
        PUSHAL  #SNALU0$K_ACTIVE        ;conn-type
        PUSHAQ  STS_DSC                 ;status-vector
    [3] PUSHAL  SESS_ID                 ;session-id

;
; Request connection
;
        CALLS   #20,G^SNALU0$REQUEST_CONNECT_W
        BLBS    R0, 130$
        BRW     EXITS
```

```
130$:
        PUSHL   #0                      ;ast parameter
        PUSHL   #0                      ;ast address
        PUSHL   #0                      ;event-flag
        PUSHL   #0                      ;sense code
        PUSHAL  #SNALU0$K_POSITIVE_RSP ;response-type
        PUSHAL  BIND_LEN                ;bind-buffer-length
        PUSHAQ  BIND_DSC                ;bind-buffer
        PUSHAQ  STS_DSC                 ;status-vector
        PUSHAL  SESS_ID                 ;session-id

;
;Acknowledge
;
        CALLS   #9,G^SNALU0$TRANSMIT_RESPONSE_W
        BLBS    R0, 140$
        BRW     EXITS

140$:
        PUSHL   #0                      ;ast parameter
        PUSHL   #0                      ;ast address
        PUSHL   #0                      ;event-flag
        PUSHAL  PLU_L_NO                ;plu-last-seq-num
        PUSHAL  PLU_F_NO                ;plu-first-seq-num
        PUSHAL  END_DATA                ;end-data
        PUSHAL  RESP_TYP                ;response-type
        PUSHAL  SENS_INC                ;sense-indicator
        PUSHAL  END_BRAC                ;end-bracket
        PUSHAL  BEG_BRAC                ;begin-bracket
        PUSHAL  ALT_CODE                ;alternate-code
        PUSHAL  FLOW                    ;message-flow
        PUSHAL  MSG_TYP                 ;message-type
        PUSHAL  MSG_CLASS               ;message-class
        PUSHAL  MORE_DATA               ;more-data-indicator
        PUSHAL  REQ_IND                 ;request/response indicator
        PUSHAL  BIND_LEN                ;bind-buffer-length
        PUSHAQ  BIND_DSC                ;bind-buffer
        PUSHAQ  STS_DSC                 ;status-vector
        PUSHAL  SESS_ID                 ;session-id
```

```
;
;Receive message
;
        CALLS   #20,G^SNALU0$RECEIVE_MESSAGE_W
        BLBS    R0, 150$
        BRW     EXITS

;
;Check whether start data traffic (SDT) received
;
150$:
        MOVL    MSG_TYP, R1
        MOVAL   SNALU0$K_MTYPE_SDT, R2
        CMPL    R1, R2
        BEQL    160$
        BRW     EXITS

160$:
        PUSHL   #0                      ;ast parameter
        PUSHL   #0                      ;ast address
        PUSHL   #0                      ;event-flag
        PUSHL   #0                      ;sense code
        PUSHAL  #SNALU0$K_POSITIVE_RSP ;response-type
        PUSHAL  BIND_LEN                ;bind-buffer-length
        PUSHAQ  BIND_DSC                ;bind-buffer
        PUSHAQ  STS_DSC                 ;status-vector
        PUSHAL  SESS_ID                 ;session-id

;
;Acknowledge SDT
;
        CALLS   #9,G^SNALU0$TRANSMIT_RESPONSE_W
        BLBS    R0, 170$
        BRW     EXITS
```

```
170$:
        PUSHL   #0                      ;ast parameter
        PUSHL   #0                      ;ast address
        PUSHL   #0                      ;event-flag
        PUSHAL  PLU_L_NO                ;plu-last-seq-num
        PUSHAL  PLU_F_NO                ;plu-first-seq-num
        PUSHAL  END_DATA                ;end-data
        PUSHAL  RESP_TYP                ;response-type
        PUSHAL  SENS_INC                ;sense-indicator
        PUSHAL  END_BRAC                ;end-bracket
        PUSHAL  BEG_BRAC                ;begin-bracket
        PUSHAL  ALT_CODE                ;alternate-code
        PUSHAL  FLOW                    ;message-flow
        PUSHAL  MSG_TYP                 ;message-type
        PUSHAL  MSG_CLASS               ;message-class
        PUSHAL  MORE_DATA               ;more-data-indicator
        PUSHAL  REQ_IND                 ;request/response indicator
        PUSHAL  BIND_LEN                ;bind-buffer-length
        PUSHAQ  BIND_DSC                ;bind-buffer
        PUSHAQ  STS_DSC                 ;status-vector
        PUSHAL  SESS_ID                 ;session-id

;
;Receive message
;
        CALLS   #20,G^SNALU0$RECEIVE_MESSAGE_W
        BLBS    R0, 180$
        BRW     EXITS

;
;Check whether BID received
;
180$:
        MOVL    MSG_TYP, R1
        MOVAL   SNALU0$K_MTYPE_BID, R2
        CMPL    R1, R2
        BEQL    190$
        BRW     EXITS
```

```
190$:
        PUSHL   #0                      ;ast parameter
        PUSHL   #0                      ;ast address
        PUSHL   #0                      ;event-flag
        PUSHL   #0                      ;sense code
        PUSHAL  #SNALU0$K_POSITIVE_RSP ;response-type
        PUSHAL  BIND_LEN                ;bind-buffer-length
        PUSHAQ  BIND_DSC                ;bind-buffer
        PUSHAQ  STS_DSC                 ;status-vector
        PUSHAL  SESS_ID                 ;session-id

;
;Acknowledge BID
;
        CALLS   #9,G^SNALU0$TRANSMIT_RESPONSE_W
        BLBS    R0, 200$
        BRW     EXITS

200$:
        PUSHL   #0                      ;ast parameter
        PUSHL   #0                      ;ast address
        PUSHL   #0                      ;event-flag
        PUSHAL  PLU_L_NO                ;plu-last-seq-num
        PUSHAL  PLU_F_NO                ;plu-first-seq-num
        PUSHAL  END_DATA                ;end-data
        PUSHAL  RESP_TYP                ;response-type
        PUSHAL  SENS_INC                ;sense-indicator
        PUSHAL  END_BRAC                ;end-bracket
        PUSHAL  BEG_BRAC                ;begin-bracket
        PUSHAL  ALT_CODE                ;alternate-code
        PUSHAL  FLOW                    ;message-flow
        PUSHAL  MSG_TYP                 ;message-type
        PUSHAL  MSG_CLASS               ;message-class
        PUSHAL  MORE_DATA               ;more-data-indicator
        PUSHAL  REQ_IND                 ;request/response indicator
        PUSHAL  BIND_LEN                ;bind-buffer-length
        PUSHAQ  BIND_DSC                ;bind-buffer
        PUSHAQ  STS_DSC                 ;status-vector
        PUSHAL  SESS_ID                 ;session-id

;
;Receive 'CICS' logo
;
        CALLS   #20,G^SNALU0$RECEIVE_MESSAGE_W
        BLBS    R0, 210$
        BRW     EXITS
```

```
210$:
        PUSHL   #0                      ;ast parameter
        PUSHL   #0                      ;ast address
        PUSHL   #0                      ;event-flag
        PUSHL   #0                      ;sense code
        PUSHAL  #SNALU0$K_POSITIVE_RSP ;response-type
        PUSHAL  BIND_LEN                ;bind-buffer-length
        PUSHAQ  BIND_DSC                ;bind-buffer
        PUSHAQ  STS_DSC                 ;status-vector
        PUSHAL  SESS_ID                 ;session-id

;
;Acknowledge 'CICS' logo
;
        CALLS   #9,G^SNALU0$TRANSMIT_RESPONSE_W
        BLBS    R0, 220$
        BRW     EXITS

220$:
;
; For this test program, the EBI or CDI should be received on this
; call. If an EBI or CDI is not received, then there is more data to
; be received. Usually, the data that has already been received is
; used to receive the rest of the data.
;
        MOVL    END_BRAC, R1
        BLBS    R1, 230$
        MOVL    END_DATA, R1
        BLBC    R1, 240$

230$:
        JMP     260$
```

```
240$:
        PUSHL   #0                      ;ast parameter
        PUSHL   #0                      ;ast address
        PUSHL   #0                      ;event-flag
        PUSHAL  PLU_L_NO                ;plu-last-seq-num
        PUSHAL  PLU_F_NO                ;plu-first-seq-num
        PUSHAL  END_DATA                ;end-data
        PUSHAL  RESP_TYP                ;response-type
        PUSHAL  SENS_INC                ;sense-indicator
        PUSHAL  END_BRAC                ;end-bracket
        PUSHAL  BEG_BRAC                ;begin-bracket
        PUSHAL  ALT_CODE                ;alternate-code
        PUSHAL  FLOW                    ;message-flow
        PUSHAL  MSG_TYP                 ;message-type
        PUSHAL  MSG_CLASS               ;message-class
        PUSHAL  MORE_DATA               ;more-data-indicator
        PUSHAL  REQ_IND                 ;request/response indicator
        PUSHAL  BIND_LEN                ;bind-buffer-length
        PUSHAQ  BIND_DSC                ;bind-buffer
        PUSHAQ  STS_DSC                 ;status-vector
        PUSHAL  SESS_ID                 ;session-id

;
;Receive rest of 'CICS' logo
;
        CALLS   #20,G^SNALU0$RECEIVE_MESSAGE_W
        BLBS    R0, 250$
        BRW     EXITS

250$:
        PUSHL   #0                      ;ast parameter
        PUSHL   #0                      ;ast address
        PUSHL   #0                      ;event-flag
        PUSHL   #0                      ;sense code
        PUSHAL  #SNALU0$K_POSITIVE_RSP ;response-type
        PUSHAL  BIND_LEN                ;bind-buffer-length
        PUSHAQ  BIND_DSC                ;bind-buffer
        PUSHAQ  STS_DSC                 ;status-vector
        PUSHAL  SESS_ID                 ;session-id

;
;Acknowledge rest of 'CICS' logo
;
        CALLS   #9,G^SNALU0$TRANSMIT_RESPONSE_W
        BLBS    R0, 260$
        BRW     EXITS
```

```
260$:
        MOVAL   BIND_BUF, R10
        MOVB    #109,B^7(R10)           ;setup clear message
        PUSHL   #0                      ;ast-parameter
        PUSHL   #0                      ;ast-address
        PUSHL   #0                      ;event-flag
        PUSHAL  SLU_L_NO                ;slu-last-seq-num
        PUSHAL  SLU_F_NO                ;slu-first-seq-num
        PUSHAL  #0                      ;turn-retain
        PUSHAL  #0                      ;more-data
        PUSHAL  #SNALU0$K_RSP_RQE1      ;response-type
        PUSHAL  #0                      ;end-bracket
        PUSHAL  #0                      ;alt-code
        PUSHAL  #SNALU0$K_MCLASS_UNFORMATTED_FM
                                        ;response-type
        PUSHAL  #8                      ;bind-buffer-length
        PUSHAQ  BIND_DSC                ;bind-buffer
        PUSHAQ  STS_DSC                 ;status-vector
        PUSHAL  SESS_ID                 ;session-id

;
;Transmit "CLEAR SCREEN"
;
        CALLS   #15,G^SNALU0$TRANSMIT_MESSAGE_W
        BLBS    R0, 270$
        BRW     EXITS

270$:
        PUSHL   #0                      ;ast parameter
        PUSHL   #0                      ;ast address
        PUSHL   #0                      ;event-flag
        PUSHAL  PLU_L_NO                ;plu-last-seq-num
        PUSHAL  PLU_F_NO                ;plu-first-seq-num
        PUSHAL  END_DATA                ;end-data
        PUSHAL  RESP_TYP                ;response-type
        PUSHAL  SENS_INC                ;sense-indicator
        PUSHAL  END_BRAC                ;end-bracket
        PUSHAL  BEG_BRAC                ;begin-bracket
        PUSHAL  ALT_CODE                ;alternate-code
        PUSHAL  FLOW                    ;message-flow
        PUSHAL  MSG_TYP                 ;message-type
        PUSHAL  MSG_CLASS               ;message-class
        PUSHAL  MORE_DATA               ;more-data-indicator
        PUSHAL  REQ_IND                 ;request/response indicator
        PUSHAL  BIND_LEN                ;bind-buffer-length
        PUSHAQ  BIND_DSC                ;bind-buffer
        PUSHAQ  STS_DSC                 ;status-vector
        PUSHAL  SESS_ID                 ;session-id
```

```
;
;Receive "CLEAR SCREEN" message
;
        CALLS   #20,G^SNALU0$RECEIVE_MESSAGE_W
        BLBS    R0, 280$
        BRW     EXITS

280$:
        PUSHL   #0                      ;ast parameter
        PUSHL   #0                      ;ast address
        PUSHL   #0                      ;event-flag
        PUSHL   #0                      ;sense code
        PUSHAL  #SNALU0$K_POSITIVE_RSP ;response-type
        PUSHAL  BIND_LEN                ;bind-buffer-length
        PUSHAQ  BIND_DSC                ;bind-buffer
        PUSHAQ  STS_DSC                 ;status-vector
        PUSHAL  SESS_ID                 ;session-id

;
;Acknowledge "CLEAR SCREEN" message
;
        CALLS   #9,G^SNALU0$TRANSMIT_RESPONSE_W
        BLBS    R0, 290$
        BRW     EXITS

290$:
        MOVAL   BIND_BUF, R10
        MOVQ    #^X00C5C6E2C3C4407D,B^7(R10)
                                        ;Transmit CSFE
        PUSHL   #0                      ;ast-parameter
        PUSHL   #0                      ;ast-address
        PUSHL   #0                      ;event-flag
        PUSHAL  SLU_L_NO                ;slu-last-seq-num
        PUSHAL  SLU_F_NO                ;slu-first-seq-num
        PUSHAL  #0                      ;turn-retain
        PUSHAL  #0                      ;more-data
        PUSHAL  #SNALU0$K_RSP_RQE1      ;response-type
        PUSHAL  #0                      ;end-bracket
        PUSHAL  #0                      ;alt-code
        PUSHAL  #SNALU0$K_MCLASS_UNFORMATTED_FM
                                        ;response-type
        PUSHAL  #14                     ;bind-buffer-length
        PUSHAQ  BIND_DSC                ;bind-buffer
        PUSHAQ  STS_DSC                 ;status-vector
        PUSHAL  SESS_ID                 ;session-id

;
;Transmit "CSFE"
;
        CALLS   #15,G^SNALU0$TRANSMIT_MESSAGE_W
        BLBS    R0, 300$
        BRW     EXITS
```

```
300$:
        PUSHL   #0                      ;ast parameter
        PUSHL   #0                      ;ast address
        PUSHL   #0                      ;event-flag
        PUSHAL  PLU_L_NO                ;plu-last-seq-num
        PUSHAL  PLU_F_NO                ;plu-first-seq-num
        PUSHAL  END_DATA                ;end-data
        PUSHAL  RESP_TYP                ;response-type
        PUSHAL  SENS_INC                ;sense-indicator
        PUSHAL  END_BRAC                ;end-bracket
        PUSHAL  BEG_BRAC                ;begin-bracket
        PUSHAL  ALT_CODE                ;alternate-code
        PUSHAL  FLOW                    ;message-flow
        PUSHAL  MSG_TYP                 ;message-type
        PUSHAL  MSG_CLASS               ;message-class
        PUSHAL  MORE_DATA               ;more-data-indicator
        PUSHAL  REQ_IND                 ;request/response indicator
        PUSHAL  BIND_LEN                ;bind-buffer-length
        PUSHAQ  BIND_DSC                ;bind-buffer
        PUSHAQ  STS_DSC                 ;status-vector
        PUSHAL  SESS_ID                 ;session-id

;
;Receive "CSFE" message
;
        CALLS   #20,G^SNALU0$RECEIVE_MESSAGE_W
        BLBS    R0, 310$
        BRW     EXITS

310$:
        PUSHL   #0                      ;ast parameter
        PUSHL   #0                      ;ast address
        PUSHL   #0                      ;event-flag
        PUSHL   #0                      ;sense code
        PUSHAL  #SNALU0$K_POSITIVE_RSP ;response-type
        PUSHAL  BIND_LEN                ;bind-buffer-length
        PUSHAQ  BIND_DSC                ;bind-buffer
        PUSHAQ  STS_DSC                 ;status-vector
        PUSHAL  SESS_ID                 ;session-id

;
;Acknowledge "CSFE" message
;
        CALLS   #9,G^SNALU0$TRANSMIT_RESPONSE_W
        BLBS    R0, 320$
        BRW     EXITS

320$:
        PUSHAQ  TEST_DSC
        PUSHAQ  TEST_SRC
```

```
;
;Translate a test buffer to transmit to the IBM
;
        CALLS   #2,G^LIB$TRA_ASC_EBC
        BLBS    R0, 330$
        BRW     EXITS

330$:
        MOVAL   TEST_BUF, R10
        MOVW    #^X7D40,B^7(R10)        ;prefix test buffer
        MOVB    #^XC4,B^9(R10)
        PUSHL   #0                      ;ast-parameter
        PUSHL   #0                      ;ast-address
        PUSHL   #0                      ;event-flag
        PUSHAL  SLU_L_NO                ;slu-last-seq-num
        PUSHAL  SLU_F_NO                ;slu-first-seq-num
        PUSHAL  #0                      ;turn-retain
        PUSHAL  #0                      ;more-data
        PUSHAL  #SNALU0$K_RSP_RQE1      ;response-type
        PUSHAL  #0                      ;end-bracket
        PUSHAL  #0                      ;alt-code
        PUSHAL  #SNALU0$K_MCLASS_UNFORMATTED_FM
                                        ;response-type
        PUSHAL  #62                     ;test-buffer-length
        PUSHAQ  TEST_DSC                ;test-buffer
        PUSHAQ  STS_DSC                 ;status-vector
        PUSHAL  SESS_ID                 ;session-id

;
;Transmit test buffer
;
        CALLS   #15,G^SNALU0$TRANSMIT_MESSAGE_W
        BLBS    R0, 340$
        BRW     EXITS
```

```
340$:
        PUSHL   #0                      ;ast parameter
        PUSHL   #0                      ;ast address
        PUSHL   #0                      ;event-flag
        PUSHAL  PLU_L_NO                ;plu-last-seq-num
        PUSHAL  PLU_F_NO                ;plu-first-seq-num
        PUSHAL  END_DATA                ;end-data
        PUSHAL  RESP_TYP                ;response-type
        PUSHAL  SENS_INC                ;sense-indicator
        PUSHAL  END_BRAC                ;end-bracket
        PUSHAL  BEG_BRAC                ;begin-bracket
        PUSHAL  ALT_CODE                ;alternate-code
        PUSHAL  FLOW                    ;message-flow
        PUSHAL  MSG_TYP                 ;message-type
        PUSHAL  MSG_CLASS               ;message-class
        PUSHAL  MORE_DATA               ;more-data-indicator
        PUSHAL  REQ_IND                 ;request/response indicator
        PUSHAL  BIND_LEN                ;bind-buffer-length
        PUSHAQ  BIND_DSC                ;bind-buffer
        PUSHAQ  STS_DSC                 ;status-vector
        PUSHAL  SESS_ID                 ;session-id

;
;Receive test message
;
        CALLS   #20,G^SNALU0$RECEIVE_MESSAGE_W
        BLBS    R0, 350$
        BRW     EXITS

350$:
        PUSHL   #0                      ;ast parameter
        PUSHL   #0                      ;ast address
        PUSHL   #0                      ;event-flag
        PUSHL   #0                      ;sense code
        PUSHAL  #SNALU0$K_POSITIVE_RSP ;response-type
        PUSHAL  BIND_LEN                ;bind-buffer-length
        PUSHAQ  BIND_DSC                ;bind-buffer
        PUSHAQ  STS_DSC                 ;status-vector
        PUSHAL  SESS_ID                 ;session-id

;
;Acknowledge test message
;
        CALLS   #9,G^SNALU0$TRANSMIT_RESPONSE_W
        BLBS    R0, 360$
        BRW     EXITS

360$:
        PUSHL   #0                      ;ast-parameter
        PUSHL   #0                      ;ast-address
        PUSHL   #0                      ;event-flag
        PUSHAQ  STS_DSC                 ;status-vector
        PUSHAL  SESS_ID                 ;session-id
```

```
;
;Disconnect
;
        CALLS   #5,G^SNALU0$REQUEST_DISCONNECT_W

EXITS:  $PUTMSG_S STS_VEC  [4]  ;display status-vector
        $EXIT_S
        .END    LU0MAR
```

**COMMENTS**

1. You must compile the API symbol definition file with your MACRO source file. For example:

   $ MACRO SYS$LIBRARY:SNALIBDEF+SNALU0DEF+SYS$DISK:[ ]*source*

2. Note that in this program the notify routine only indicates that the application received the event. Normally, the application would take some action.
3. Arguments are passed on the stack.
4. Display the status vector by using $PUTMSG.

## E.5 VAX PL/I Programming Example

This program connects to CICS and activates the CSFE transaction (a remote loopback program). The program prompts you for the Gateway DECnet node or TCP/IP host name and the CICS access name. The application then establishes a connection with CICS and requests the CSFE transaction. After the CSFE instruction screen is received, some test data is generated. The data is converted to EBCDIC and sent to CSFE. The data is echoed from CSFE and the session is disconnected.

```
  MAIN: PROCEDURE OPTIONS(MAIN) RETURNS (FIXED BINARY(31));

  %INCLUDE $STSDEF;                        /* System status codes       */
  %INCLUDE SYS$PUTMSG;                     /* System Service            */
[1]%INCLUDE 'SYS$LIBRARY:SNALU0DEF.PLI';   /* LU0 symbols and           */
                                           /*  routine definitions      */
[1]%INCLUDE 'SYS$LIBRARY:SNALIBDEF.PLI';   /* Basic AI symbols and      */
                                           /*  routine definitions      */

  /*******************************************************************/
  /* Declare External Routines first                                 */
  /*******************************************************************/
  DECLARE
      EXAMPLE$NOTIFY POINTER GLOBALREF,

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

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

  /********************************************************************/
  /* Declarations                                                     */
  /********************************************************************/
  %REPLACE CSFE_SIZE BY 14;
  %REPLACE TEST_SIZE BY 62;
  %REPLACE BUFFER_SIZE BY 4103;
  %REPLACE SESSION_ADDRESS BY 0;
```

```
DECLARE
    NODE_NAME CHARACTER (6),
    NODE_NAME_SIZE FIXED BIN (15),
    NODE_PROMPT CHARACTER (26) STATIC INITIAL('Enter gateway node: '),
    ACCESS_NAME CHARACTER (6),
    ACCESS_NAME_SIZE FIXED BIN (15),
    ACCESS_PROMPT CHARACTER (20) STATIC INITIAL('Enter access name: '),
    SESSION_ID FIXED BIN (31),
    DATA_SIZE FIXED BIN (15),
    BIND_SIZE FIXED BIN (15),
    MSG_TYPE FIXED BIN (7),
    MSG_CLASS FIXED BIN (7),
    MSG_EBI FIXED BIN (7),
    FIRST_SEQNO FIXED BIN (15),
    LAST_SEQNO FIXED BIN (15),
    STATUS_VECTOR CHARACTER (SNALU0$K_MIN_STATUS_VECTOR),
    NOTIFY_VECTOR GLOBALDEF CHARACTER (SNALU0$K_MIN_NOTIFY_VECTOR),
    CSFE_TEXT CHARACTER (CSFE_SIZE) INITIAL('         CSFE'),
    DATA_BUFFER_PTR POINTER,
    DATA_BUFFER CHARACTER (BUFFER_SIZE),
    DATA_BUFFER_ARRAY (BUFFER_SIZE) CHARACTER BASED (DATA_BUFFER_PTR),
    TEST_MESSAGE CHARACTER (TEST_SIZE) INITIAL
        ('          ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz');

DATA_BUFFER_PTR = ADDR(DATA_BUFFER);

STS$VALUE = EXAMPLE$MAIN();
STOP;
/*********************************************************************/
/* The END                                                           */
/*********************************************************************/

EXAMPLE$MAIN: PROCEDURE RETURNS (FIXED BINARY(31));

/*********************************************************************/
/* Get node name and access name. Use this information               */
/* to establish a session with IBM.                                  */
/*********************************************************************/
STS$VALUE = LIB$GET_INPUT(      NODE_NAME,
                                NODE_PROMPT,
                                NODE_NAME_SIZE);
IF ^STS$SUCCESS THEN RETURN(STS$VALUE);

STS$VALUE = LIB$GET_INPUT(      ACCESS_NAME,
                                ACCESS_PROMPT,
                                ACCESS_NAME_SIZE);
IF ^STS$SUCCESS THEN RETURN(STS$VALUE);
```

```
STS$VALUE = SNALU0$REQUEST_CONNECT_W(
                                SESSION_ID,
                                STATUS_VECTOR,
                                SNALU0$K_ACTIVE,
                                NODE_NAME,
                                ACCESS_NAME,
                                ,
                                SESSION_ADDRESS,
                              [2] ,,,,,
                                ADDR(EXAMPLE$NOTIFY),
                                ,
                                NOTIFY_VECTOR,
                                DATA_BUFFER,
                                BIND_SIZE,
                                ,,);
IF ^STS$SUCCESS THEN GOTO EXIT;

/*******************************************************************/
/* Assume the BIND is OK and just send +RSP to it                  */
/*******************************************************************/
STS$VALUE = SNALU0$TRANSMIT_RESPONSE_W(
                                SESSION_ID,
                                STATUS_VECTOR,
                                DATA_BUFFER,
                                DATA_SIZE,
                                SNALU0$K_POSITIVE_RSP,
                                ,,,);

IF ^STS$SUCCESS THEN GOTO EXIT;

/*******************************************************************/
/* Issue three receives to get :                                   */
/*                                                                 */
/*  Start Data Traffic (SDT)                                       */
/*  BID                                                            */
/*  CICS logo                                                      */
/*******************************************************************/
STS$VALUE = EXAMPLE$READ_DATA( );
IF ^STS$SUCCESS THEN GOTO EXIT;
STS$VALUE = EXAMPLE$READ_DATA( );
IF ^STS$SUCCESS THEN GOTO EXIT;
STS$VALUE = EXAMPLE$READ_DATA( );
IF ^STS$SUCCESS THEN GOTO EXIT;

/*******************************************************************/
/* Transmit a CSFE message to test the session and get the         */
/* response message.                                               */
/*******************************************************************/
[3]STS$VALUE = LIB$TRA_ASC_EBC(
                        CSFE_TEXT,
                        CSFE_TEXT);
IF ^STS$SUCCESS THEN GOTO EXIT;
```

```
[4]DATA_BUFFER = CSFE_TEXT;
  DATA_BUFFER_ARRAY(SNABUF$K_HDLEN+1) = BYTE(125);        /*  AID       */
  DATA_BUFFER_ARRAY(SNABUF$K_HDLEN+2) = BYTE(64);         /*Cursor      */
[4]DATA_BUFFER_ARRAY(SNABUF$K_HDLEN+3) = BYTE(196);        /*Address     */
  FIRST_SEQNO = LAST_SEQNO + 1;
  LAST_SEQNO = FIRST_SEQNO;              /* Single element chain      */

  STS$VALUE = SNALU0$TRANSMIT_MESSAGE_W(
                        SESSION_ID,
                        STATUS_VECTOR,
                        DATA_BUFFER,
                        CSFE_SIZE,
                        SNALU0$K_MCLASS_UNFORMATTED_FM,
                        ,,
                        SNALU0$K_RSP_RQE1,
                        ,,
                        FIRST_SEQNO,
                        LAST_SEQNO,
                        ,,);
  IF ^STS$SUCCESS THEN GOTO EXIT;

  STS$VALUE = EXAMPLE$READ_DATA( );
  IF ^STS$SUCCESS THEN GOTO EXIT;

  /*******************************************************************/
  /* Transmit a test message to test the session and get the         */
  /* response message.                                               */
  /*******************************************************************/
  STS$VALUE = LIB$TRA_ASC_EBC(
                        TEST_MESSAGE,
                        TEST_MESSAGE);
  IF ^STS$SUCCESS THEN GOTO EXIT;

  DATA_BUFFER = TEST_MESSAGE;
  DATA_BUFFER_ARRAY(SNABUF$K_HDLEN+1) = BYTE(125);       /*  AID       */
  DATA_BUFFER_ARRAY(SNABUF$K_HDLEN+2) = BYTE(64);        /*Cursor      */
  DATA_BUFFER_ARRAY(SNABUF$K_HDLEN+3) = BYTE(196);       /*Addres      */
  FIRST_SEQNO = LAST_SEQNO + 1;
  LAST_SEQNO = FIRST_SEQNO;              /* Single element chain      */

  STS$VALUE = SNALU0$TRANSMIT_MESSAGE_W(
                        SESSION_ID,
                        STATUS_VECTOR,
                        DATA_BUFFER,
                        TEST_SIZE,
                        SNALU0$K_MCLASS_UNFORMATTED_FM,
                        ,,
                        SNALU0$K_RSP_RQE1,
                        ,,
                        FIRST_SEQNO,
                        LAST_SEQNO,
                        ,,);
  IF ^STS$SUCCESS THEN GOTO EXIT;
```

```
STS$VALUE = EXAMPLE$READ_DATA( );
IF ^STS$SUCCESS THEN GOTO EXIT;

STS$VALUE = SNALU0$REQUEST_DISCONNECT_W(
                        SESSION_ID,
                        STATUS_VECTOR);

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


/*********************************************************************/
/* Receive data and send a +RSP                                      */
/*********************************************************************/
EXAMPLE$READ_DATA:      PROCEDURE RETURNS (FIXED BIN);

%INCLUDE $STSDEF;                       /* System status codes       */
%INCLUDE 'SYS$LIBRARY:SNALU0DEF.PLI';   /* SNALU0 symbols and        */
                                        /*  routine definitions      */
DECLARE
    RRI FIXED BINARY (7),
    MORE_DATA FIXED BINARY (7),
    RCV_FLOW FIXED BINARY (7),
    RCV_CODE FIXED BINARY (7),
    RCV_BBI FIXED BINARY (7),
    RCV_SDI FIXED BINARY (7),
    RCV_RTYPE FIXED BINARY (7),
    RCV_CDI FIXED BINARY (7);

STS$VALUE = SNALU0$RECEIVE_MESSAGE_W(
                                SESSION_ID,
                                STATUS_VECTOR,
                                DATA_BUFFER,
                                DATA_SIZE,
                                RRI,
                                MORE_DATA,
                                MSG_CLASS,
                                MSG_TYPE,
                                RCV_FLOW,
                                RCV_CODE,
                                RCV_BBI,
                                MSG_EBI,
                                RCV_SDI,
                                RCV_RTYPE,
                                RCV_CDI,
                                FIRST_SEQNO,
                                LAST_SEQNO,
                                ,,);
```

```
IF STS$SUCCESS
THEN
    STS$VALUE = SNALU0$TRANSMIT_RESPONSE_W(
                                SESSION_ID,
                                STATUS_VECTOR,
                                DATA_BUFFER,
                                DATA_SIZE,
                                SNALU0$K_POSITIVE_RSP,
                                ,,,);

/*********************************************************************/
/* For this test program, the EBI or CDI should be received on this   */
/* call. If an EBI or CDI is not received, then there is more data    */
/* to be received. Usually, the data that has already been received   */
/* is used to receive the rest of the data.                           */
/*********************************************************************/
IF STS$SUCCESS & ^(MSG_EBI = 1) & ^(RCV_CDI = 1) &
   ^(POSINT(MSG_TYPE) = SNALU0$K_MTYPE_SDT) &
   ^(POSINT(MSG_TYPE) = SNALU0$K_MTYPE_BID)
THEN
    BEGIN;
    STS$VALUE = SNALU0$RECEIVE_MESSAGE_W(
                                    SESSION_ID,
                                    STATUS_VECTOR,
                                    DATA_BUFFER,
                                    DATA_SIZE,
                                    RRI,
                                    MORE_DATA,
                                    MSG_CLASS,
                                    MSG_TYPE,
                                    RCV_FLOW,
                                    RCV_CODE,
                                    RCV_BBI,
                                    MSG_EBI,
                                    RCV_SDI,
                                    RCV_RTYPE,
                                    RCV_CDI,
                                    FIRST_SEQNO,
                                    LAST_SEQNO,
                                    ,,);

    IF STS$SUCCESS
    THEN
        STS$VALUE = SNALU0$TRANSMIT_RESPONSE_W(
                                    SESSION_ID,
                                    STATUS_VECTOR,
                                    DATA_BUFFER,
                                    DATA_SIZE,
                                    SNALU0$K_POSITIVE_RSP,
                                    ,,,);
    END;
RETURN(STS$VALUE);
```

```
END;
END;

/******************************************************************/
/* Asynchronous Notify Routine                                    */
/******************************************************************/
EXAMPLE$NOTIFY: PROCEDURE(  EVENT_CODE,
                            EVENT_PARAMETER);

%INCLUDE $STSDEF;                      /* System status codes      */
%INCLUDE SYS$PUTMSG;                   /* System Service           */
%INCLUDE 'SYS$LIBRARY:SNALU0DEF.PLI';  /* SNA3270 symbols and routine*/
                                       /* definitions              */
%INCLUDE 'SYS$LIBRARY:SNALIBDEF.PLI';  /* Basic AI symbols and     */
                                       /*  routine definitions     */

/******************************************************************/
/* Declare External Routines first                                */
/******************************************************************/
DECLARE
    LIB$PUT_OUTPUT EXTERNAL ENTRY (
            CHARACTER (*))                     /* Data              */
    RETURNS (FIXED BIN(31));

/******************************************************************/
/* Declarations                                                   */
/******************************************************************/
DECLARE
    EVENT_CODE FIXED BINARY (31),
    EVENT_PARAMETER FIXED BINARY (31),
    NOTIFY_VECTOR GLOBALREF CHARACTER,
    RCVEXP_EVENT CHARACTER (58) STATIC INITIAL
        ('Asynchronous notification: Data receive on expedited flow'),
    COMERR_EVENT CHARACTER (54) STATIC INITIAL
        ('Asynchronous notification: Gateway communication error'),
    TERM_EVENT CHARACTER (57) STATIC INITIAL
        ('Asynchronous notification: Session terminated by IBM host'),
    PLURESET_EVENT CHARACTER (60) STATIC INITIAL
        ('Asynchronous notification: Half session state machines reset'),
    UNBHLD_EVENT CHARACTER (49) STATIC INITIAL
        ('Asynchronous notification: Unbind type 2 received'),
    UNKNOWN_EVENT CHARACTER (62) STATIC INITIAL
        ('Asynchronous notification: Unknown asynchronous event reported');

SELECT (EVENT_CODE);
    WHEN (SNAEVT$K_RCVEXP)
        STS$VALUE = LIB$PUT_OUTPUT( RCVEXP_EVENT );
    WHEN (SNAEVT$K_UNBHLD)
        STS$VALUE = LIB$PUT_OUTPUT( UNBHLD_EVENT );
    WHEN (SNAEVT$K_TERM)
        STS$VALUE = LIB$PUT_OUTPUT( TERM_EVENT );
    WHEN (SNAEVT$K_COMERR)
        STS$VALUE = LIB$PUT_OUTPUT( COMERR_EVENT );
```

```
    WHEN (SNAEVT$K_PLURESET)
        STS$VALUE = LIB$PUT_OUTPUT( PLURESET_EVENT );
    OTHERWISE
        STS$VALUE = LIB$PUT_OUTPUT( UNKNOWN_EVENT );
END;
STS$VALUE = SYS$PUTMSG(NOTIFY_VECTOR); [6]
END;
```

**COMMENTS**

1. Include the LU0 and basic API symbol definition libraries.
2. Commas indicate that you do not want to specify values for the parameters and will accept the default values provided by the API.
3. You can use OpenVMS Library routines to do parts of your application, such as translating ASCII to EBCDIC or vice versa.
4. The application must leave room in the buffer for header information.
5. The asynchronous notify routine, notify parameter, and notify vector are specified in the REQUEST_CONNECT procedure. If you are using multiple sessions, specify *session-id* or an internal session data structure in the *event-parameter*, so you can identify a particular session. For more information, see Section 3.6.
6. Display the NOTIFY_VECTOR by using $PUTMSG.

## E.6 Pascal Programming Example

This program connects to CICS and activates the CSFE transaction (a remote loopback program). It prompts you for the Gateway DECnet node or TCP/IP host name and the CICS access name. The application then establishes a connection with CICS and requests the CSFE transaction. After the CSFE instruction screen is received, some test data is generated. The data is converted to EBCDIC and sent to CSFE. The data is echoed from CSFE and the session is disconnected.

```
[1] [INHERIT ('SYS$LIBRARY:SNALU0DEF.PEN', 'SYS$LIBRARY:SNALIBDEF.PEN')]
PROGRAM   LU0_EXAMPLE(INPUT,OUTPUT);

(*      %INCLUDE 'SNALU0DEF/NOLIST'     *)
(*      %INCLUDE 'SNALIBDEF/NOLIST'     *)

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

[ASYNCHRONOUS,EXTERNAL(SYS$PUTMSG)] FUNCTION $PUTMSG
                        (%REF MSGVEC : [UNSAFE] ARRAY
                           [$l1..$u1:INTEGER] OF $UBYTE;
                        %IMMED [UNBOUND, ASYNCHRONOUS]
                           PROCEDURE ACTRTN := %IMMED 0;
                        FACNAM : [CLASS_S] PACKED ARRAY
                           [$l3..$u3:INTEGER] OF CHAR
                           := %IMMED 0;
                        %IMMED ACTPRM : INTEGER := %IMMED 0)
                        : INTEGER; EXTERNAL;

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

[ASYNCHRONOUS] FUNCTION LIB$GET_INPUT
                        (%STDESCR IN_BUFFER: CHAR): INTEGER;
                        EXTERNAL;
```

```
[ASYNCHRONOUS] FUNCTION LIB$STOP
                        (COND_VAL: UNSIGNED): INTEGER;
                        EXTERNAL;

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

    LABEL
        999;

    CONST
        BUFFER_LENGTH   = 4103;
        LU0_EFN         = 10;
        SESSION_ADDRESS = 0;

    VAR
        ACCESS_NAME     : VARYING[8] OF CHAR;
        ASCII_CSFE_STR  : PACKED ARRAY [1..4] OF CHAR;
        ASCII_TEST_DATA : PACKED ARRAY [1..52] OF CHAR;
        CANNED_MESSAGE  : PACKED ARRAY [1..28] OF CHAR;
        CSFE_STR        : PACKED ARRAY [1..4] OF CHAR;
    [2] BUFFER_HEADER   : PACKED ARRAY [1..SNABUF$K_HDLEN] OF CHAR;
        DATA_BUFFER     : PACKED ARRAY [1..BUFFER_LENGTH] OF CHAR;
    [2] DS_HEADER       : PACKED ARRAY [1..3] OF CHAR;
        NODE_NAME       : VARYING[8] OF CHAR;
        NOTIFY_MESSAGE  : PACKED ARRAY [1..28] OF CHAR;
        NOTIFY_VECTOR   : PACKED ARRAY [1..SNALU0$K_MIN_NOTIFY_VECTOR] OF CHAR;
        STATUS_VECTOR   : PACKED ARRAY [1..SNALU0$K_MIN_STATUS_VECTOR] OF CHAR;
        TEST_DATA       : PACKED ARRAY [1..52] OF CHAR;
        LIB_STATUS      : UNSIGNED;
        SNA_STATUS      : UNSIGNED;
        EVENT_CODE      : INTEGER;
        SESSION_ID      : INTEGER;
        DATA_LENGTH     : $WORD;
        PLU_1ST_SEQ_NUM : $WORD;
        PLU_LAST_SEQ_NUM: $WORD;
        SLU_1ST_SEQ_NUM : $WORD;
        SLU_LAST_SEQ_NUM: $WORD;
        ALT_CODE        : $BYTE;
        BEG_BRACKET     : $BYTE;
        END_BRACKET     : $BYTE;
        END_DATA        : $BYTE;
        FLOW            : $BYTE;
        MORE_DATA       : $BYTE;
        MSG_CLASS       : $BYTE;
        MSG_TYPE        : $BYTE;
        REQ_IND         : $BYTE;
        RESP_TYPE       : $BYTE;
        SENSE_INC       : $BYTE;

FUNCTION XMIT_POS_RESP: BOOLEAN;
```

```
    BEGIN
        SNA_STATUS := SNALU0$TRANSMIT_RESPONSE_W(SESSION_ID,
                                                 STATUS_VECTOR,
                                                 DATA_BUFFER,
                                                 DATA_LENGTH,
                                                 %REF(SNALU0$K_POSITIVE_RSP));

        IF SNA_STATUS :: BOOLEAN
        THEN
            XMIT_POS_RESP := TRUE
        ELSE
            BEGIN
                LIB_STATUS := $PUTMSG(STATUS_VECTOR);
                WRITELN('TRANSMIT_RESPONSE failed');
                XMIT_POS_RESP := FALSE;
            END
    END;

FUNCTION RECEIVE : BOOLEAN;
    BEGIN
        SNA_STATUS := SNALU0$RECEIVE_MESSAGE_W(SESSION_ID,
                                               STATUS_VECTOR,
                                               DATA_BUFFER,
                                               DATA_LENGTH,
                                               REQ_IND,
                                               MORE_DATA,
                                               MSG_CLASS,
                                               MSG_TYPE,
                                               FLOW,
                                               ALT_CODE,
                                               BEG_BRACKET,
                                               END_BRACKET,
                                               SENSE_INC,
                                               RESP_TYPE,
                                               END_DATA,
                                               PLU_1ST_SEQ_NUM,
                                               PLU_LAST_SEQ_NUM,
                                               %REF(LU0_EFN));
```

```
      [3] IF SNA_STATUS :: BOOLEAN
      [3] THEN
            RECEIVE := TRUE
        ELSE
            BEGIN
                LIB_STATUS := $PUTMSG(STATUS_VECTOR);
                WRITELN('RECEIVE_MESSAGE failed');
                RECEIVE := FALSE
            END;
    END;

FUNCTION RECEIVE_SDT : BOOLEAN;
    BEGIN
        RECEIVE_SDT := RECEIVE;                 (* Receive start data traffic *)
        IF (MSG_TYPE <> SNALU0$K_MTYPE_SDT)
        THEN
            WRITELN('Failed to receive SDT')
        ELSE
            RECEIVE_SDT := XMIT_POS_RESP;       (* Acknowledge SDT *)
    END;

FUNCTION RECEIVE_DATA : BOOLEAN;
    BEGIN
        IF RECEIVE                              (* Receive data or BID *)
        THEN
            IF (MSG_TYPE = SNALU0$K_MTYPE_BID)  (* Check for BID, if received,*)
            THEN                                (* acknowledge it and issue a *)
                IF XMIT_POS_RESP                (* receive to get CICS logo *)
                THEN
                    RECEIVE_DATA := RECEIVE
                ELSE
                    RECEIVE_DATA := FALSE
            ELSE
                RECEIVE_DATA := TRUE
        ELSE
            RECEIVE_DATA := FALSE;
    END;                                        (* Return with data or error *)

FUNCTION TRANSMIT(LENGTH : INTEGER;
                  BUFFER : PACKED ARRAY [LB..UB : INTEGER] OF CHAR) : BOOLEAN;
    BEGIN
        SNA_STATUS := SNALU0$TRANSMIT_MESSAGE_W(SESSION_ID,
                                                STATUS_VECTOR,
                                                BUFFER,
                                                LENGTH,
                                                MSG_CLASS,
                                                %REF(FALSE),
                                                %REF(FALSE),
                                                %REF(SNALU0$K_RSP_RQE1),
                                                %REF(FALSE),
                                                %REF(FALSE),
                                                SLU_1ST_SEQ_NUM,
```

```
                                                SLU_LAST_SEQ_NUM,
                                                %REF(LU0_EFN));
        IF SNA_STATUS :: BOOLEAN
        THEN
            TRANSMIT := TRUE
        ELSE
            BEGIN
                LIB_STATUS := $PUTMSG(STATUS_VECTOR);
                WRITELN('TRANSMIT_MESSAGE failed')
            END;
    END;

PROCEDURE NOTIFY_RTN(EVENT_CODE, NOTIFY_PARM : INTEGER);
    BEGIN
        WRITE('Asynchronous notification: ');
        CASE EVENT_CODE OF
            SNAEVT$K_RCVEXP:
                WRITELN('Data received on expedited flow');
            SNAEVT$K_UNBHLD:
                WRITELN('Unbind type 2 received');
            SNAEVT$K_TERM:
                WRITELN('Session terminated by IBM host');
            SNAEVT$K_COMERR:
                WRITELN('Gateway communication error');
            SNAEVT$K_PLURESET:
                WRITELN('Half session state machines reset');
            OTHERWISE
                WRITELN('Unknown asynchronous event reported');
        END;
        LIB_STATUS := $PUTMSG(NOTIFY_VECTOR);
    END;
```

```
    BEGIN
        BUFFER_HEADER := '        ';
        DATA_BUFFER := ' ';
        WRITE('Enter gateway node:');
        READLN(NODE_NAME);
        WRITE('Enter access name: ');
        READLN(ACCESS_NAME);

(*************************************************************************)
(* Request connection                                                    *)
(*************************************************************************)
        SNA_STATUS := SNALU0$REQUEST_CONNECT_W(SESSION_ID,
                                               STATUS_VECTOR,
                                               %REF(SNALU0$K_ACTIVE),
                                               NODE_NAME,
                                               ACCESS_NAME,,
                                               %REF(SESSION_ADDRESS),
                                               ,,,,,
                                               %IMMED NOTIFY_RTN,
                                               SESSION_ID,
                                               NOTIFY_VECTOR,
                                               DATA_BUFFER,
                                               DATA_LENGTH,
                                               %REF(LU0_EFN));

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

(***********************************************************************)
(*                                                                     *)
(* Acknowledge Bind - in this example we assume the BIND is            *)
(* satisfactory. Normally you would have to examine the bind image     *)
(* to verify that your application can handle the session defined by  *)
(* the BIND.                                                           *)
(*                                                                     *)
(* A negotiable BIND would be effected by modifying the BIND image     *)
(* and positively responding.                                          *)
(*                                                                     *)
(***********************************************************************)

        IF (NOT XMIT_POS_RESP)
        THEN
            GOTO 999;
```

```
(*******************************************************************)
(* Receive start data traffic                                      *)
(*******************************************************************)
        IF (NOT RECEIVE_SDT)
        THEN
            BEGIN
                WRITELN('Failed to receive Start Data Traffic');
                GOTO 999
            END;

(*******************************************************************)
(* Receive CICS logo                                               *)
(*******************************************************************)
        IF (NOT RECEIVE_DATA)
        THEN
            BEGIN
                WRITELN('Failed to receive CICS logo');
                GOTO 999
            END;

(*******************************************************************)
(* Acknowledge CICS logo                                           *)
(*******************************************************************)
        IF (NOT XMIT_POS_RESP)
        THEN
            GOTO 999;

(*******************************************************************)
(* For this test program, the EBI or CDI should be received on this   *)
(* call. If an EBI or CDI is not received, then there is more data    *)
(* to be received. Usually, the data that has already been received   *)
(* is used to receive the rest of the data.                           *)
(*******************************************************************)
        IF (NOT (END_BRACKET = 1) AND NOT(END_DATA = 1))
        THEN
            BEGIN
            IF (NOT RECEIVE_DATA)
            THEN
                BEGIN
                    WRITELN('Failed to receive CICS logo');
                    GOTO 999
                END;
            IF (NOT XMIT_POS_RESP)
            THEN
                GOTO 999;
            END;
```

```
(*******************************************************************)
(* Send clear screen request                                       *)
(*******************************************************************)
        DATA_LENGTH := SNABUF$K_HDLEN+1;

(*******************************************************************)
(* Set EBCDIC code for clear                                       *)
(*******************************************************************)
        DATA_BUFFER[8] := CHR(%X'6D');
        MSG_CLASS := SNALU0$K_MCLASS_UNFORMATTED_FM;
        IF (NOT TRANSMIT(DATA_LENGTH, DATA_BUFFER))
        THEN
            BEGIN
                WRITELN('Failed to transmit clear screen request');
                GOTO 999
            END;

(*******************************************************************)
(* Receive clear screen command                                    *)
(*******************************************************************)
        IF (NOT RECEIVE_DATA)
        THEN
            BEGIN
                WRITELN('Failed to receive clear screen command');
                GOTO 999
            END;

(*******************************************************************)
(* Acknowledge clear screen command                                *)
(*******************************************************************)
        IF (NOT XMIT_POS_RESP)
        THEN
            GOTO 999;

(*******************************************************************)
(* Transmit CSFE string and 3270 data stream control characters (3).  *)
(*******************************************************************)
        ASCII_CSFE_STR := 'CSFE';
```

```
(**********************************************************************)
(* Translate CSFE string to EBCDIC                                    *)
(**********************************************************************)
        LIB_STATUS := LIB$TRA_ASC_EBC(ASCII_CSFE_STR, CSFE_STR);
        IF (NOT LIB_STATUS :: BOOLEAN)
        THEN
            LIB$STOP(LIB_STATUS);

(**********************************************************************)
(* Insert the 3 3270 control characters AID key = ENTER, Cursor       *)
(* position = 04 (encoded)                                            *)
(**********************************************************************)
        DS_HEADER := ''(%X'7D',%X'40',%X'C4')'';
        MSG_CLASS := SNALU0$K_MCLASS_UNFORMATTED_FM;
        DATA_LENGTH := SNABUF$K_HDLEN + 3 + 4;
        IF (NOT TRANSMIT(DATA_LENGTH,
                         (BUFFER_HEADER + DS_HEADER + CSFE_STR)))
        THEN
            BEGIN
                WRITELN('Failed to transmit the CSFE string');
                GOTO 999
            END;

(**********************************************************************)
(* Receive CSFE message                                               *)
(**********************************************************************)
        IF (NOT RECEIVE_DATA)
        THEN
            BEGIN
                WRITELN('Failed to receive CSFE message');
                GOTO 999
            END;

(**********************************************************************)
(* Acknowledge CSFE message                                           *)
(**********************************************************************)
        IF (NOT XMIT_POS_RESP)
        THEN
            GOTO 999;

(**********************************************************************)
(* Transmit test message                                              *)
(**********************************************************************)
        ASCII_TEST_DATA :=
            'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz';
```

```
(*******************************************************************)
(* Translate test message to EBCDIC                               *)
(*******************************************************************)
        LIB_STATUS := LIB$TRA_ASC_EBC(ASCII_TEST_DATA, TEST_DATA);
        IF (NOT LIB_STATUS :: BOOLEAN)
        THEN
            LIB$STOP(LIB_STATUS);

(*******************************************************************)
(* Insert the 3 3270 control characters                           *)
(* AID key = ENTER, Cursor position = 04 (encoded)                *)
(*******************************************************************)
        DS_HEADER := ''(%X'7D',%X'40',%X'C4')'';
        MSG_CLASS := SNALU0$K_MCLASS_UNFORMATTED_FM;
        DATA_LENGTH := SNABUF$K_HDLEN+55;
        IF (NOT TRANSMIT(DATA_LENGTH,
                         (BUFFER_HEADER + DS_HEADER + TEST_DATA)))
        THEN
            BEGIN
                WRITELN('Failed to transmit the TEST string');
                GOTO 999
            END;

(*******************************************************************)
(* Receive TEST message                                            *)
(*******************************************************************)
        IF (NOT RECEIVE_DATA)
        THEN
            BEGIN
                WRITELN('Failed to receive TEST message');
                GOTO 999
            END;

(*******************************************************************)
(* Acknowledge TEST message                                        *)
(*******************************************************************)
        IF (NOT XMIT_POS_RESP)
        THEN
            GOTO 999;

(*******************************************************************)
(* Disconnect session                                              *)
(*******************************************************************)
        SNA_STATUS := SNALU0$REQUEST_DISCONNECT_W(SESSION_ID,
                                                  STATUS_VECTOR);

999:
        LIB_STATUS := $PUTMSG(STATUS_VECTOR);

    END.
```

**COMMENTS**

1. Include the LU0 and basic API symbol definition libraries.
2. Three buffers are defined for the 3270 data stream message:
   - Data buffer
   - Buffer header (to reserve space for the API)
   - Data stream header (to reserve 3 bytes for 3270 data stream control characters)

   The buffers are concatenated by means of the Pascal string operators.
3. Use the type cast operator to override the declared type for the returned status.

## E.7 Pascal Symbol and Structure Definitions

The API supplies symbol and structure definitions used by the application program. For Pascal, these are provided in source code files SNALU0DEF.PAS and SNALIBDEF.PAS, and in environmental files SNALU0DEF.PEN and SNALIBDEF.PEN. These environmental files must be generated using the PASCAL compiler on your OpenVMS system. To generate the environmental files, enter the following commands:

```
$ PASCAL/NOOBJECT/ENVIRONMENT=SYS$LIBRARY:SNALIBDEF.PEN -
$_ SYS$LIBRARY:SNALIBDEF.PAS
$ PASCAL/NOOBJECT/ENVIRONMENT=SYS$LIBRARY:SNALU0DEF.PEN -
$_ SYS$LIBRARY:SNALU0DEF.PAS_FOR_PEN
```

## E.8 C Programming Example

This program connects to CICS and activates the CSFE transaction (a remote loopback program). It prompts you for the Gateway DECnet node or TCP/IP host name and the CICS access name. The application then establishes a connection with CICS and requests the CSFE transaction. After the CSFE instruction screen is received, some test data is generated. The data is converted to EBCDIC and sent to CSFE. The data is echoed from CSFE and the session is disconnected.

```
#module LU0_TEST

[1] #include <SNALU0DEF>
#include descrip                        /* VMS descriptor definitions*/
#include stsdef                         /* Define STS$type_name      */

#define S_SIZE snalu0$k_min_status_vector
#define N_SIZE snalu0$k_min_notify_vector

/*******************************************************************/
/* Declaration                                                     */
/*******************************************************************/
        globaldef int   STATUS_VEC[S_SIZE],NOTIFY_VEC[N_SIZE],SID,RCV_COUNT;
        globaldef int   RET,NOTIFY_PARM,SENSE_CODE;
        globaldef struct dsc$descriptor   BUF_D;

        $DESCRIPTOR(STATUS_D, STATUS_VEC);
        $DESCRIPTOR(NOTIFY_D, NOTIFY_VEC);
main()
{
extern CONNECT(),RECEIVE(),TRANSMIT();

    RET=CONNECT();      /* Get the BIND and acknowledge it           */
    if (!(RET & STS$M_SUCCESS))
    {
        return(RET);
    }

    RET=RECEIVE();      /* Receive SDT and send +RSP                 */
    if (!(RET & STS$M_SUCCESS))
    {
        return(RET);
    }
```

```
    RET=RECEIVE();      /* Receive BID and send +RSP                */
    if (!(RET & STS$M_SUCCESS))
    {
        return(RET);
    }

    RET=RECEIVE();      /* Receive CICS logo and send +RSP          */
    if (!(RET & STS$M_SUCCESS))
    {
        return(RET);
    }

    RET=TRANSMIT();     /* Transmit and receive some data           */
    if (!(RET & STS$M_SUCCESS))
    {
        return(RET);
    }
/*********************************************************************/
/*      Disconnect the session and exit with status                 */
/*********************************************************************/
    RET= SNALU0$REQUEST_DISCONNECT_W(
                                &SID,
                                &STATUS_D
                                );
    SYS$PUTMSG(STATUS_VEC);
}

/*********************************************************************/
/*                                                                   */
/* Get the node name and access name to use in establishing a        */
/* session. Issue a SNALU0$REQUEST_CONNECT_W to get the BIND. Send   */
/* a +RSP to the BIND and return.                                    */
/*                                                                   */
/*********************************************************************/

#include <SNALU0DEF>                    /* external definition file  */
#include descrip                        /* VMS descriptor definitions*/
#include stsdef                         /* Define STS$type_name      */

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

globalref int   SID,RET,SENSE_CODE,STATUS_VEC[];
globalref struct dsc$descriptor   BUF_D;

CONNECT()
{
    extern NOTIFY_RTN(), BIND_CHK();
    extern STATUS_D, NOTIFY_D;
```

```
    int                 BIND_LENGTH, SESSION_ADDRESS;
    char        NODE_NAME[8],ACC_NAME[8];
    struct      dsc$descriptor   NODE_D,ACC_D;

/*******************************************************************/
/* Initialize the descriptors we are going to use                  */
/*******************************************************************/
        NODE_D.dsc$b_dtype = DSC$K_DTYPE_T;
        NODE_D.dsc$b_class = DSC$K_CLASS_S;
        NODE_D.dsc$a_pointer = NODE_NAME;

        ACC_D.dsc$b_dtype = DSC$K_DTYPE_T;
        ACC_D.dsc$b_class = DSC$K_CLASS_S;
        ACC_D.dsc$a_pointer = ACC_NAME;

        BUF_D.dsc$w_length = 0;
        BUF_D.dsc$b_dtype = DSC$K_DTYPE_T;
        BUF_D.dsc$b_class = DSC$K_CLASS_D;
        BUF_D.dsc$a_pointer = 0;
        SESSION_ADDRESS = 0;

/*******************************************************************/
/* Get the node name and access name to use in establishing a      */
/* session with IBM.                                               */
/*******************************************************************/
        printf("Enter gateway node name:");
        scanf("%s",NODE_NAME);

        printf("Enter acess name:");
        scanf("%s",ACC_NAME);

        NODE_D.dsc$w_length = strlen(NODE_NAME);
        ACC_D.dsc$w_length = strlen(ACC_NAME);

/*******************************************************************/
/* Bring up the session with the IBM system. On error output the   */
/* status vector and return.                                       */
/*******************************************************************/
        RET = SNALU0$REQUEST_CONNECT_W(
                                &SID,
                                &STATUS_D,
                                &snalu0$k_active,
                                &NODE_D,
                                &ACC_D,
                                0,
                                &SESSION_ADDRESS,
                            [2] 0,0,0,0,0,
                                NOTIFY_RTN,
                                &SID,
                                &NOTIFY_D,
                                &BUF_D,
                                &BIND_LENGTH
                                );
```

```
        if (!(RET & STS$M_SUCCESS))
        {
                SYS$PUTMSG(STATUS_VEC);
                return(RET);
        }

/*******************************************************************/
/*                                                                 */
/* Acknowledge Bind - in this example we assume the BIND is        */
/* satisfactory. Normally you would have to examine the bind image  */
/* to verify that your application can handle the session defined by */
/* the BIND.                                                       */
/*                                                                 */
/* A negotiable BIND would be effected by modifying the BIND image  */
/* and positively responding.                                      */
/*                                                                 */
/*******************************************************************/
        RET=SNALU0$TRANSMIT_RESPONSE_W(
                                &SID,
                                &STATUS_D,
                                &BUF_D,
                                &BIND_LENGTH,
                                &snalu0$k_positive_rsp
                                );

        LIB$SFREE1_DD(&BUF_D);          /* Free up the dynamic buffer
                                           we have been using       */
        if (!(RET & STS$M_SUCCESS))
        {
                SYS$PUTMSG(STATUS_VEC);
        }
        return(RET);
}

#include <SNALU0DEF>                    /* Definition file          */
#include descrip                        /* VMS descriptor definitions*/
#include stsdef                         /* Status definition file    */
#define  RL 4103                        /* Define data buffer size   */

/*******************************************************************/
/* Declaration                                                     */
/*******************************************************************/
    globalref int       SID,RET,STATUS_VEC[];
    globalref struct    dsc$descriptor   BUF_D;

    extern STATUS_D;
    int           RCV_COUNT,RCV_EFN;
    int           BUFFER_LENGTH,FIRST_SQN,LAST_SQN;
    unsigned char RID,MORE,MSG_CLASS,MSG_TYPE,FLOW_TYPE,ALT_CODE,BBI,EBI;
    unsigned char SENSE,RESP_TYPE,CDI,RCV_BUF[RL];

RECEIVE()
{
```

```
/*********************************************************************/
/*  Initialize descriptors and put up a receive                     */
/*********************************************************************/
    BUF_D.dsc$w_length = RL;
    BUF_D.dsc$b_dtype = DSC$K_DTYPE_T;
    BUF_D.dsc$b_class = DSC$K_CLASS_D;
    BUF_D.dsc$a_pointer = RCV_BUF;

    RET=SNALU0$RECEIVE_MESSAGE_W(
                            &SID,
                            &STATUS_D,
                            &BUF_D,
                            &BUFFER_LENGTH,
                            &RID,
                            &MORE,
                            &MSG_CLASS,
                            &MSG_TYPE,
                            &FLOW_TYPE,
                            &ALT_CODE,
                            &BBI,
                            &EBI,
                            &SENSE,
                            &RESP_TYPE,
                            &CDI,
                            &FIRST_SQN,
                            &LAST_SQN,
                            0,
                            0,
                            0
                            );

    if (!(RET & STS$M_SUCCESS))
    {
        SYS$PUTMSG(STATUS_VEC);
        return(RET);
    }

/*********************************************************************/
/*      If receive was successful send a +RSP                       */
/*********************************************************************/
        RET = SNALU0$TRANSMIT_RESPONSE_W(
                            &SID,
                            &STATUS_D,
                            &BUF_D,
                            &BUFFER_LENGTH,
                            &snalu0$k_positive_rsp,
                            0,
                            0,
                            0,
                            0
                            );
```

```
    if (!(RET & STS$M_SUCCESS))
    {
        SYS$PUTMSG(STATUS_VEC);
        return(RET);
    }

/*******************************************************************/
/* For this test program, the EBI or CDI should be received on this  */
/* call. If an EBI or CDI is not received, then there is more data   */
/* to be received. Usually, the data that has already been received  */
/* is used to receive the rest of the data.                          */
/*******************************************************************/
    if (!(EBI & STS$M_SUCCESS) && !(CDI & STS$M_SUCCESS) &&
        (MSG_TYPE != snalu0$k_mtype_sdt) &&
        (MSG_TYPE != snalu0$k_mtype_bid))
    {
        RET=SNALU0$RECEIVE_MESSAGE_W(
                                &SID,
                                &STATUS_D,
                                &BUF_D,
                                &BUFFER_LENGTH,
                                &RID,
                                &MORE,
                                &MSG_CLASS,
                                &MSG_TYPE,
                                &FLOW_TYPE,
                                &ALT_CODE,
                                &BBI,
                                &EBI,
                                &SENSE,
                                &RESP_TYPE,
                                &CDI,
                                &FIRST_SQN,
                                &LAST_SQN,
                                0,
                                0,
                                0
                                );
```

```
        if (!(RET & STS$M_SUCCESS))
        {
            SYS$PUTMSG(STATUS_VEC);
            return(RET);
        }

/*********************************************************************/
/*      If receive was successful send a +RSP                        */
/*********************************************************************/
            RET = SNALU0$TRANSMIT_RESPONSE_W(
                                &SID,
                                &STATUS_D,
                                &BUF_D,
                                &BUFFER_LENGTH,
                                &snalu0$k_positive_rsp,
                                0,
                                0,
                                0,
                                0
                                );

        if (!(RET & STS$M_SUCCESS))
        {
            SYS$PUTMSG(STATUS_VEC);
        }
    }

return(RET);
}

#include    <SNALU0DEF>                 /* external definition file  */
#include    <SNALIBDEF>                 /* external definition file  */
#include    descrip                     /* VMS descriptor definitions*/
#include    stsdef                      /* Define STS$type_name      */
#define     CLEAR_LEN 8
#define     TEST_LEN 62
#define     CSFE_LEN 14

/*********************************************************************/
/* Declaration                                                       */
/*********************************************************************/
globalref int    SID, RET, SENSE_CODE, STATUS_VEC[];

TRANSMIT()
{
    extern                      STATUS_D, RECEIVE();
    int                         BIND_LENGTH, SLU_FIRST_SEQ, SLU_LAST_SEQ;
    int                         I;
    char                        TEST_BUF[TEST_LEN];
    struct dsc$descriptor       TEST_D;
```

```
/*******************************************************************/
/*      Initialize the descriptor we are going to use              */
/*******************************************************************/
    TEST_D.dsc$b_class = DSC$K_CLASS_S;
    TEST_D.dsc$b_dtype = DSC$K_DTYPE_T;
    TEST_D.dsc$w_length = TEST_LEN;
    TEST_D.dsc$a_pointer = TEST_BUF;
    for (I = 0;  I < 62;  I = I + 1)
    {
        TEST_BUF[I] = 0;
    }

/*******************************************************************/
/*      Send clear screen request                                  */
/*******************************************************************/
    TEST_BUF[SNABUF$K_HDLEN] = 109;

    RET = SNALU0$TRANSMIT_MESSAGE_W(
                            &SID,
                            &STATUS_D,
                            &TEST_D,
                            &CLEAR_LEN,
                            &snalu0$k_mclass_unformatted_fm,
                            0,0,
                            &snalu0$k_rsp_rqe1,
                            0,0,
                            &SLU_FIRST_SEQ,
                            &SLU_LAST_SEQ,
                            0,0,0
                            );

    if (!(RET & STS$M_SUCCESS))
    {
        SYS$PUTMSG(STATUS_VEC);
        return(RET);
    }

    RET=RECEIVE();      /* Get clear message back and send +RSP       */
    if (!(RET & STS$M_SUCCESS))
    {
        return(RET);
    }

/*******************************************************************/
/*      Fill the test buffer with the characters "CSFE"            */
/*******************************************************************/
    TEST_BUF[SNABUF$K_HDLEN] = 0;
    TEST_BUF[SNABUF$K_HDLEN + 3] = 67;
    TEST_BUF[SNABUF$K_HDLEN + 4] = 83;
    TEST_BUF[SNABUF$K_HDLEN + 5] = 70;
    TEST_BUF[SNABUF$K_HDLEN + 6] = 69;
```

```
    RET = LIB$TRA_ASC_EBC( &TEST_D, &TEST_D );
    if (!(RET & STS$M_SUCCESS))
    {
        SYS$PUTMSG(STATUS_VEC);
        return(RET);
    }

    TEST_BUF[SNABUF$K_HDLEN] = 125;
    TEST_BUF[SNABUF$K_HDLEN + 1] = 64;
    TEST_BUF[SNABUF$K_HDLEN + 2] = 196;

    RET = SNALU0$TRANSMIT_MESSAGE_W(
                        &SID,
                        &STATUS_D,
                        &TEST_D,
                        &CSFE_LEN,
                        &snalu0$k_mclass_unformatted_fm,
                        0,0,
                        &snalu0$k_rsp_rqe1,
                        0,0,
                        &SLU_FIRST_SEQ,
                        &SLU_LAST_SEQ,
                        0,0,0
                        );

    if (!(RET & STS$M_SUCCESS))
    {
        SYS$PUTMSG(STATUS_VEC);
        return(RET);
    }
    RET=RECEIVE();      /* Get message back and send +RSP           */
    if (!(RET & STS$M_SUCCESS))
    {
        return(RET);
    }

/*********************************************************************/
/*      Send a test message                                          */
/*********************************************************************/
    TEST_BUF[SNABUF$K_HDLEN] = 0;
    TEST_BUF[SNABUF$K_HDLEN + 1] = 0;
    TEST_BUF[SNABUF$K_HDLEN + 2] = 0;
    for (I = 0;  I < 52;  I = I + 1)
    {
        if (I < 26)
            TEST_BUF[SNABUF$K_HDLEN + 3 + I] = 65 + I;
        else
            TEST_BUF[SNABUF$K_HDLEN + 3 + I] = 97 + I - 26;
    }
```

```
    RET = LIB$TRA_ASC_EBC( &TEST_D, &TEST_D );
    if (!(RET & STS$M_SUCCESS))
    {
        SYS$PUTMSG(STATUS_VEC);
        return(RET);
    }

    TEST_BUF[SNABUF$K_HDLEN] = 125;
    TEST_BUF[SNABUF$K_HDLEN + 1] = 64;
    TEST_BUF[SNABUF$K_HDLEN + 2] = 196;

    RET = SNALU0$TRANSMIT_MESSAGE_W(
                            &SID,
                            &STATUS_D,
                            &TEST_D,
                            &TEST_LEN,
                            &snalu0$k_mclass_unformatted_fm,
                            0,0,
                            &snalu0$k_rsp_rqe1,
                            0,0,
                            &SLU_FIRST_SEQ,
                            &SLU_LAST_SEQ,
                            0,0,0
                            );

    if (!(RET & STS$M_SUCCESS))
    {
        SYS$PUTMSG(STATUS_VEC);
        return(RET);
    }

    RET=RECEIVE();              /* Get message back and send +RSP    */
    return(RET);
}

#include <SNALIBDEF>    /* Basic AI source                           */

/*********************************************************************/
/* Declaration                                                       */
/*********************************************************************/
globalref int NOTIFY_VEC[],STATUS_VEC[];
```

```
NOTIFY_RTN(EVENT_CODE, P_SESSION_ID)
int EVENT_CODE,P_SESSION_ID;
{
        if (EVENT_CODE == SNAEVT$K_RCVEXP)
           printf("Asynchronous notification: Data received on expedited flow");
        else if (EVENT_CODE == SNAEVT$K_UNBHLD)
           printf("Asynchronous notification: Unbind type 2 received");
        else if (EVENT_CODE == SNAEVT$K_TERM)
           printf("Asynchronous notification: Session terminated by IBM host");
        else if (EVENT_CODE == SNAEVT$K_COMERR)
           printf("Asynchronous notification: Gateway communication error");
        else if (EVENT_CODE == SNAEVT$K_PLURESET)
           printf("Asynchronous notification: Half session state machines
           reset");
        else
           printf("Asynchronous notification: Unknown asynchronous event
           reported");

    [3] SYS$PUTMSG(NOTIFY_VEC);
}
```

**COMMENTS**

1. Include the LU0 and basic API symbol definition libraries.
2. The commas and zeros indicate that you do not want to specify values for the parameters and will accept the default values provided by the API.
3. Display the NOTIFY_VECTOR by using $PUTMSG.

# F
# Status Codes

Status messages that are returned by the Digital SNA Application Programming Interface (API) for OpenVMS, as well as the lower layers of SNA software, are displayed at your terminal in the following format:

where

*facility* is the SNALU0 or the SNA component name. A percent sign (%) prefixes the first message displayed on your screen, and a hyphen prefixes each subsequent message. SNALU0 refers to those messages received from the extended mode of the API and SNA refers to those messages received from the basic mode.

*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 the 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:

```
%SNALU0-E-EVTCLR, failed to clear an event flag
```

The following status messages are returned by the API:

ABNSESTER, session terminated abnormally

**Facility:** SNA

**Explanation:** Either the link between the SNA Gateway and IBM was lost or IBM deactivated the physical unit or the line leading to the SNA Gateway.

**User Action:** Determine why the link was lost. Retry when the connection with IBM returns.

ACCINTERR, Gateway detected an error in the Gateway access routines

**Facility:** SNA

**Explanation:** This is a fatal error.

**User Action:** Copy the error messages that appear on your screen at this time and report the problem to your system manager.

ACCROUFAI, error from Gateway access routine, gateway unknown or unreachable

**Facility:** SNA

**Explanation:** SNA Gateway is unknown or unreachable; Transport list (defined by SNA_TRANSPORT_ORDER logical) is defined incorrectly or SNA Gateway specified does not support transport selected; or TCP/IP Port (defined by SNA_TCP_PORT logical) does not match the remote connection TCP/IP Port.

**User Action:** Check the SNA Gateway, the SNA_TRANSPORT_ORDER logical, or the SNA_TCP_PORT logical.

ACCTOOLON, access name is too long

**Facility:** SNA

**Explanation:** The access name must be no longer than 8 characters.

**User Action:** Correct the call to the API procedure.

ACQLU, unable to acquire a logical unit

**Facility:** SNALU0

**Explanation:** The API failed to obtain a port into the SNA network.

**User Action:** Examine the status vector for further information.

AEFOUTRAN, asynchronous event flag number is out of range

**Facility:** SNA

**Explanation:** The event flag number specified was greater than 127.

**User Action:** Correct the call to the API procedure.

AEVTER, error processing an async event

**Facility:** SNALU0

**Explanation:** Insufficient resources were available for the API while in an internal AST routine.

**User Action:** Write down all the messages associated with this error and report the problem to your system manager.

APPNOTSPE, IBM application name was not specified

**Facility:** SNA

**Explanation:** You did not specify the IBM application name in the SNA$CONNECT call, and the access name that you used did not supply one either.

**User Action:** Either explicitly supply the IBM application in the parameter list or implicitly supply it through the access name.

APPTOOLON, application name is too long

**Facility:** SNA

**Explanation:** The application name must be no longer than 8 characters.

**User Action:** Correct the call to the API procedure.

BINSPEUNA, the BIND image specified unacceptable values

**Facility:** SNA

**Explanation:** The SNA Gateway rejected the BIND request image.

**User Action:** Run a trace to find out why the SNA Gateway rejected the BIND request. The IBM application could be specifying too large an outbound RU or an illegal FM or TS profile, or it could have sent a pacing value that was out of bounds (see the *Digital SNA Guide to IBM Parameters*).

BUFTOOSHO, transmit buffer is too short

**Facility:** SNA

**Explanation:** The transmit buffer must be at least SNABUF$K_HDLEN bytes long.

**User Action:** You probably forgot to leave SNABUF$K_HDLEN bytes at the front of your transmit buffer for API use.

BUGCHK, internal error detected in *LU0 module*

**Facility:** SNALU0

**Explanation:** A fatal error has occurred.

**User Action:** Write down all the messages associated with this error and report the problem to your system manager.

CONREQREJ, connect request rejected by IBM host, sense code %X'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.

DATTOOLON, too much user data specified

**Facility:** SNA

**Explanation:** The user data you can specify must be limited to 128 bytes.

**User Action:** Correct the call to the API procedure.

DISCFAIL, call to SNALU0$REQUEST_DISCONNECT[_W] failed

**Facility:** SNALU0

**Explanation:** The call to this procedure failed.

**User Action:** Examine the status vector for further information.

EVFOUTRAN, event flag number is out of range

**Facility:** SNA

**Explanation:** The event flag number specified was greater than 127.

**User Action:** Correct the call to use a valid event flag number.

EVTCLR, failed to clear an event flag

**Facility:** SNALU0

**Explanation:** The event flag parameter could not be cleared.

**User Action:** Make sure that a valid event flag was specified.

EXIT, Gateway server task terminated

**Facility:** SNALU0

**Explanation:** The SNA Gateway access server has exited abnormally.

**User Action:** See the *Digital SNA Gateway Problem Determination Guide* or the *OpenVMS/SNA Problem Determination Guide* for additional information. If you still cannot solve your problem, notify the system manager.

FAIALLBUF, failed to allocate memory for a buffer

**Facility:** SNA

**Explanation:** The API 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 with LIB$SFREE1_DD or STR$FREE1_DX.

FAIALLCTX, failed to allocate memory for a context block

**Facility:** SNA

**Explanation:** The API 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 with 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 API failed to build a DECnet network connect block in order to communicate with the SNA Gateway.

**User Action:** Examine the error code in the second longword of the I/O status block (IOSB) for more information.

FAICONMBX, failed to convert mailbox name

**Facility:** SNA

**Explanation:** The API 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 API could not copy the entire BIND request image into the BIND request buffer provided by the application.

**User Action:** Examine the error code in the second longword of the IOSB for more information. Make sure that you specify a BIND buffer large enough to receive the largest BIND that the IBM application will send you.

FAICOPBUF, failed to copy data into caller's buffer

**Facility:** SNA

**Explanation:** The API could not copy all of the received RU into the buffer provided by the application.

**User Action:** Examine the error code in the second longword of the IOSB for more information. Make sure that you specify a data buffer that is large enough to receive the largest data RU that the IBM application will send you.

FAIESTLIN, failed to establish a link to the Gateway

**Facility:** SNA

**Explanation:** The API cannot connect to the SNA Gateway.

**User Action:** Examine the subsequent DECnet error messages and take appropriate action.

FATINTERR, internal error in Gateway access routines

**Facility:** SNA

**Explanation:** This is a fatal error.

**User Action:** Write down all the messages that appear on your screen at this time and report the problem to your system manager.

FUNCABORT, access routine function aborted

**Facility:** SNA

**Explanation:** The API 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 API is invalid with the port in the current state. You issued API 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:** SNA

**Explanation:** There was an error in communicating with the SNA Gateway node.

**User Action:** Examine the DECnet error message in the second longword of the IOSB and take appropriate action.

GATINTERR, internal error in Gateway node, code %O'xx',subcode %O'xx'

**Facility:** SNA

**Explanation:** A fatal error has occurred.

**User Action:** Report the error to your system manager. Also ensure that the log from the SNA Gateway console is saved the log will have messages of the form

GAS—Fatal Session Error FSE$xxx

GETLU0VM, failed to obtain memory for LU 0 processing

**Facility:** SNALU0

**Explanation:** The API was unable to obtain virtual memory for internal data structures.

**User Action:** Make sure that the application is returning memory space that it has finished using. If you are using class D descriptors, make sure you return used buffers to free memory with LIB$SFREE1_DD or STR$SFREE_DX.

ILLASTSTA, ASTs are disabled or an AST routine is currently in progress

**Facility:** SNA

**Explanation:** A call was made to an API 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, no action has been taken by the procedure.

**User Action:** Change the application so that API 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 SNA Gateway is incompatible with the SNA software on the local system.

**User Action:** Make sure that the correct versions of the software are installed on both the SNA Gateway and the local system.

INSGATRES, insufficient Gateway resources for session establishment

**Facility:** SNA

**Explanation:** The SNA Gateway has insufficient resources for establishing a session. The active sessions currently in the SNA 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 API could not allocate enough system resources to establish the session.

**User Action:** Examine the second longword in the IOSB for more information.

INVBUF, invalid buffer

**Facility:** SNALU0

**Explanation:** The application supplied an invalid buffer.

**User Action:** Examine the status vector for additional information.

INVFMBIND, BIND specified an invalid FM profile of 19

**Facility:** SNALU0

**Explanation:** The BIND received was rejected because it specified an FM profile of 19.

**User Action:** Check to see that your transaction is not specifying an access name that could send this type of BIND. Consult your IBM system manager if the correct BIND is being sent.

INVRECLOG, SNA$DEF_NUMREC is incorrectly defined

**Facility:** SNA

**Explanation:** This internal logical name is set up improperly.

**User Action:** SNA$DEF_NUMREC is a logical name that determines the number of receives the API keeps outstanding on the DECnet logical link. If you do not wish to use the default value, use the DEFINE command (for example, DEFINE SNA$DEF_NUMREC 5).

INVSESID, invalid session ID supplied

**Facility:** SNALU0

**Explanation:** The session ID passed by the application is not a session ID returned by the SNALU0$REQUEST_CONNECT procedure.

**User Action:** Verify that the session is still active. Supply the correct session ID.

LENTOOLON, data length is too long

**Facility:** SNA

**Explanation:** The DATALEN parameter to SNA$TRANSMIT specified a message length longer than the length field in the transmit buffer descriptor parameter.

**User Action:** Correct the call to specify actual length of the data you wish to transmit.

LMTTOOLON, logon mode name is too long

**Facility:** SNA

**Explanation:** The logon mode name must be no longer than 8 characters.

**User Action:** Correct the call to the API procedure.

LOGUNIDEA, SSCP has deactivated the session

**Facility:** SNA

**Explanation:** The IBM SSCP has deactivated the session by sending a DACTLU command. Some applications deactivate sessions by deactivating the LU rather than by sending an UNBIND command.

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 SNA$TERMINATE procedure for each session that has terminated.

MUTOEXR, MU was converted to an exception request/response

**Facility:** SNA

**Explanation:** The message unit was converted to an exception request /response.

MUTORCVCHK, MU generated a receive check, sense code %X'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 %X'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.

NETSHUT, network node is not accepting connects

**Facility:** SNALU0

**Explanation:** The SNA Gateway has been told to shut down.

**User Action:** Try again later when a SNA Gateway is activated.

NO_GWYNOD, SNA$DEF_GATEWAY is undefined and GWY-NODE was not specified

**Facility:** SNA

**Explanation:** A SNA Gateway node was not specified in the SNA$CONNECT or SNA$LISTEN call, and the logical name SNA$DEF_GATEWAY was not defined.

**User Action:** Either supply an explicit SNA 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 PU.

**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.

NOEVEPEN, no event pending

**Facility:** SNA

**Explanation:** You issued an SNA$READEVENT and there were no outstanding events to read for the session.

**User Action:** Wait until the API notifies you of an asynchronous event for the session and then issue another SNA$READEVENT.

NOSESN, failed to start session

**Facility:** SNALU0

**Explanation:** The call to the SNALU0$REQUEST_CONNECT[_W] procedure failed to start a session with the PLU.

**User Action:** Examine the status vector for further information.

NOTVECTSM, notify vector is too small

**Facility:** SNALU0

**Explanation:** The notify vector is too small to contain the largest notify message.

**User Action:** Make sure that notify vector contains at least SNALU0$K_ MIN_NOTIFY_VECTOR bytes.

PARERR, parameter error, routine SNALU0$*xxx*[_W]

**Facility:** SNALU0

**Explanation:** The application supplied a bad parameter for the SNALU0$*xxx*[_W] procedure.

**User Action:** Examine the status vector to determine the bad parameter and change it.

PASTOOLON, password is too long

**Facility:** SNA

**Explanation:** The IBM password must be no longer than 8 characters.

**User Action:** Correct the call to the API procedure.

PLUPROVIO, PLU violated SNA protocol rules, sense code %X'IBM sense code'

**Facility:** SNA

**Explanation:** The PLU violated SNA protocol rules.

**User Action:** Consult your IBM manual for the sense code.

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.

PUNOTAVA, PU has not been activated

**Facility:** SNA

**Explanation:** The PU on the SNA Gateway has not been activated by IBM.

**User Action:** Ask the VTAM operator to check the line and physical unit from the IBM host and activate them if necessary. If they are activated, there may be a hardware problem between the SNA Gateway and the IBM host.

PUNOTSPE, PU name was not specified

**Facility:** SNA

**Explanation:** You did not specify a PU name in the SNA$CONNECT or the SNA$LISTEN call, and the access name that you used did not supply one either.

**User Action:** Either explicitly supply the PU name in the parameter list or implicitly supply it through the access name.

PUTOOLON, PU name is too long

**Facility:** SNA

**Explanation:** The PU name must be no longer than 8 characters.

**User Action:** Correct the call to the API procedure.

RCNFAIL, call to SNALU0$REQUEST_RECONNECT[_W] failed

**Facility:** SNALU0

**Explanation:** The call to this procedure failed.

**User Action:** Examine the status vector for further information.

RCVBFSM, receive buffer is too small, must be *xxx* bytes while *yyy* were supplied

**Facility:** SNALU0

**Explanation:** The receive buffer for the SNALU0$REQUEST_CONNECT, SNALU0$REQUEST_RECONNECT, or SNALU0$RECEIVE_MESSAGE is too small.

**User Action:** For a REQUEST_CONNECT or REQUEST_RECONNECT call, make the buffer size equal to or greater than SNABUF$K_LENGTH. For a RECEIEVE_MESSAGE call, make the buffer size equal to SNABUF$K_HDLEN plus the maximum outbound (that is, from PLU to OpenVMS application) RU size for the session.

RCVFAIL, call to receive failed

**Facility:** SNALU0

**Explanation:** The attempt to receive a message from the PLU failed.

**User Action:** Examine the status vector for further information.

RECTOOLAR, receive count is too large

**Facility:** SNA

**Explanation:** You have specified more than 10 outstanding receives.

**User Action:** Correct the SNA$CONNECT or SNA$LISTEN call to specify *numrec* to be 10 or less, or omit the parameter so that the API uses the default of 3 outstanding receives.

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.

SESINUNAC, session address already in use or not activated

**Facility:** SNA

**Explanation:** All session addresses in the range specified by the access name are in use or are not activated.

**User Action:** Ask the IBM VTAM operator to activate more SLUs, or wait for an active one to become available.

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 LU from the IBM host and activate it if necessary.

TERMPEND, SNA$TERMINATE has already been issued

**Facility:** SNA

**Explanation:** You have called SNA$TERMINATE more than once for the session.

**User Action:** Ignore the error or correct the logical error in your application.

TONEGRSP, failed to change RU to negative response

**Facility:** SNALU0

**Explanation:** The API was unable to convert the buffer supplied by the application into a negative response.

**User Action:** Make sure that the buffer header reserved for the API has not been modified.

TOOFEWPAR, not enough parameters specified

**Facility:** SNA

**Explanation:** An API procedure was called with too few parameters.

**User Action:** Supply the required parameters.

TOOMANPAR, too many parameters specified

**Facility:** SNA

**Explanation:** An API procedure was called with too many parameters.

**User Action:** Use only the specified parameters when calling a procedure.

TOPOSRSP, failed to change RU to positive response

**Facility:** SNALU0

**Explanation:** The API was unable to convert the buffer supplied by the application into a positive response.

**User Action:** Make sure that the buffer header reserved for the API has not been modified.

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.

UNALEF, unable to get an internal local event flag

**Facility:** SNALU0

**Explanation:** The API was unable to obtain an event flag for its internal use.

**User Action:** Make sure the OpenVMS application calls LIB$FREE_EF to free any temporary event flags that it uses.

UNBINDREC, UNBIND request received from IBM application

**Facility:** SNA

**Explanation:** The IBM application has terminated the session by sending a normal UNBIND RU.

UNUUNBREC, UNBIND of type %X'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 code from the IBM documentation on the UNBIND request and take the appropriate action.

USETOOLON, user name is too long

**Facility:** SNA

**Explanation:** The user name must be no longer than 8 characters.

**User Action:** Correct the call to the API procedure.

XMTFAIL, call to SNALU0$TRANSMIT[_W] failed

**Facility:** SNALU0

**Explanation:** The call to SNALU0$TRANSMIT_MESSAGE[_W] or SNALU0$TRANSMIT_RESPONSE[_W] failed.

**User Action:** Examine the status vector for further information.

# G

# Correlation of Procedures and Status Messages for the API

Figure G–1 illustrates the correlation between API procedures and the status messages they can return.

**Figure G–1 Correlation of Procedures and Status Messages for the API**

| Status message | Examine state | Receive message | Request connect | Request reconnect | Request disconnect | Transmit message | Transmit response |
|---|---|---|---|---|---|---|---|
| SNALU0$_ACQLU | | | X | | | | |
| SNALU0$_DISCFAIL | | | | | X | | |
| SNALU0$_EVTCLR | | X | X | X | X | X | X |
| SNALU0$_GETLU0VM | | X | X | X | X | X | X |
| SNALU0$_INVBUF | | X | X | X | | X | X |
| SNALU0$_INVSESID | | X | | X | X | X | X |
| SNALU0$_NORMAL | X | X | X | X | X | X | X |
| SNALU0$_NOSESN | | | X | | | | |
| SNALU0$_PARERR | | X | X | X | X | X | X |
| SNALU0$_RCNFAIL | | | | X | | | |
| SNALU0$_RCVBFSM | | X | X | X | | | |
| SNALU0$_RCVFAIL | | X | X | | | | |
| SNALU0$_TONEGRSP | | | | | | | X |
| SNALU0$_TOPOSRS | | | | | | | X |
| SNALU0$_UNALEF | | | | X | | | |
| SNALU0$_XMTFAIL | | | | | | X | X |
| SNALU0$_FREELU | | | | | X | | |
| SNALU0$_FREELU0VM | | | | | | | X |
| SNALU0$_ILEFWT | | X | X | X | X | X | X |

LKG–8104–93R

# Index


## A

Access information, IBM, 3–8
  access name, 3–9
  Gateway Physical Unit, 3–8
  OpenVMS/SNA PU identification, 3–8
  optional user data, 3–8
  password, 3–8
  PLU application name, 3–8
  SLU session address, 3–8
  user identification, 3–8
Access information, IMB
  logon mode name, 3–8
Access name, 3–9
Active connect request
  issuing, 2–2
  issuing a typical active connect request, 2–4
API procedures
  correlation with status messages, G–1
  SNALU0$EXAMINE_STATE, 5–2
  SNALU0$RECEIVE_MESSAGE, 5–4
  SNALU0$REQUEST_CONNECT, 5–8
  SNALU0$REQUEST_DISCONNECT, 5–12
  SNALU0$REQUEST_RECONNECT, 5–11
  SNALU0$TRANSMIT_MESSAGE, 5–13
  SNALU0$TRANSMIT_RESPONSE, 5–17
API symbol definitions, D–1
Arguments
  passing, 5–2
Asynchronous event notification, 3–12
Asynchronous mode operation, 3–6
  completion procedure, use of, 3–6
  event flag, use of, 3–6

## B

BID request, 4–10
BIND request, 4–13, B–1
  accepting or rejecting, 2–6
Bracketing requests, 4–10
Brackets, 2–8, 4–7
  bracket termination rule 1, 4–7
  bracket termination rule 2, 4–7

## C

C programming example, E–59
CANCEL request, 4–8
Chaining, 2–8, 4–4
  chaining indicators, 2–8, 4–4
  control modes, 4–5
  request modes, 4–6
  response modes, 4–6
CHASE request, 4–10
Cleanup requests, 4–9
CLEAR request, 4–13
COBOL programming example, E–15
Communication functions
  function management (FM) profile, 2–1
  transmission subsystem (TS) profile, 2–1
Compiling a transaction program, 6–1
Completion procedure
  use in asynchronous mode operation, 3–6

Connection point manager (CPMGR), 4–11

## D

Data flow control requests, 4–8
Definite responses, 4–5
Descriptors, use of, 2–15

## E

Event flag
  use by API procedures, 3–7
  use in asynchronous mode operation, 3–6
Exception responses, 4–5

## F

FORTRAN Definition Files, E–14
FORTRAN programming example, E–1
Function management (FM) profiles, 4–3
Function value returns, 3–1

## G

Gateway PU identification, 3–8

## I

IBM password, 3–8
IBM user identification, 3–8

## L

Linking
  application, with shareable image, 6–2
  transaction program, 6–1
Logon mode name, 3–8
LU network services, 4–1
LU session control functions, 4–12
LU status (LUSTAT) request, 4–10
LU-LU type 0 session, 2–1
  establishing a session, 2–2
    issuing a passive connect request, 2–4
    issuing an active connect request, 2–2
LU-LU type 0 session (cont'd)
  reestablishing, 2–18
  terminating, 2–18

## M

MACRO programming example, E–24
Message classes and types, 3–9

## N

No response chains, 4–5

## O

OpenVMS/SNA PU identification, 3–8

## P

Pacing, 4–12
Pascal programming example, E–48
Pascal Symbol and Structure Definitions, E–58
Passive connect request
  issuing, 2–4
  issuing a typical passive connect request, 2–5
Pause requests, 4–9
PLU application name, 3–8
Presentation services, 4–2
Procedure parameter notation, A–1
Procedures
  See API procedures
Programming examples
  C, E–59
  COBOL, E–15
  FORTRAN, E–1
  MACRO, E–24
  Pascal, E–48
  VAX PL/I, E–40
Programming Examples, E–1

## Q


Quiesce at end of chain (QEC) request, 4–9
Quiesce complete (QC) request, 4–9


## R


Ready to receive (RTR) request, 4–10
Release quiesce (RELQ) request, 4–9
Request header, 2–7
Request modes, 4–6
Request recover (RQR) request, 4–14
Request shutdown (RSHUTD) request, 4–10
Request unit chains
  transmission of control information, 2–13
  transmission of user data, 2–10
Request unit classes and types
  table, 3–9
Request units
  bracketing, 2–8
  chains as, 2–8
  change direction indicator (CDI), 2–8
  request header, 2–7
  sending, 2–7
  sequence numbers, 2–9
  unique identifiers, 2–9
Request/response header (RH), C–4
  building, 4–11
Request/response mode protocols, 4–6
Request/response unit (RU), 2–7
  receiving, 2–15
Response modes, 4–6
Response types, 4–5
  definite response chains, 4–5
  exception response chains, 4–5
  no response chains, 4–5
Response units
  transmitting, 2–14


## S


Secondary logical unit (SLU)
  with Digital SNA Gateway, 2–2
Send/receive modes, 4–3
  duplex mode, 4–4
  half duplex contention mode, 4–4
  half duplex flip-flop mode, 4–3
Sequence number field (SNF), 4–11
Session, LU-LU
  establishing, 2–2
    issuing a passive connect request, 2–4
    issuing an active connect request, 2–2
  reestablishing, 2–18
  terminating, 2–18
Set and test sequence number (STSN) request, 4–14
Shutdown (SHUTD) request, 4–10
Shutdown complete (SHUTC) request, 4–10
SIGNAL request, 4–10
SLU session address, 3–8
SNA layers, 4–1
  Data Flow Control layer, 4–3
  LU services layer, 4–1
  Transmission Control layer, 4–11
Start data traffic (SDT) request, 4–13
Status codes, 3–1, F–1
  function value returns, 3–1
  status vector, 3–2
Status messages
  correlation with API procedures, G–1
Status vector
  and $PUTMSG, 3–2
  illustration, 3–4
  using, 3–2
Synchronous mode operation, 3–5

## T

Transmission of user data, 2–10
    simplest type, 2–12
Transmission subsystem (TS) profiles, 4–11
Transmitting control information, 2–13

## U

UNBIND request, 4–13

## V

VAX PL/I programming example, E–40

## W

Wait mode operation, 3–5