Digital SNA 3270 Data Stream Programming Interface for OpenVMS

Programming

Part Number: AA–ET84E–TE

May 1996

This document supplies information about the services provided by the Digital SNA 3270 Data Stream Programming Interface that enable an OpenVMS application to exchange messages with a cooperating application on the 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 3270 Data Stream Programming Interface for OpenVMS, Version 1.5

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.

Contents

3 Interface Features

4 Linking an Application With the 3270 Data Stream Interface

5 Programming Examples

Part III Reference

6 Procedure Calling Formats

Part IV Appendixes

A Attention Identification Values

B Summary of Procedure Parameter Notation

C Definitions for the 3270 Data Stream Programming Interface

D Status Codes

E Low-Level Status Codes

F Correlation of Procedures and Status Messages for the 3270 Data Stream

Index

Figures

Tables

Preface

The Digital SNA 3270 Data Stream Programming Interface for OpenVMS is a Digital Equipment Corporation software product. It enables OpenVMS VAX users or Alpha systems users running OpenVMS SNA (OpenVMS VAX Version 6.1 and Version 6.2 only) with programs running on an IBM host or connected to either of the following SNA Gateways:

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

The Interface allows you to develop applications on an OpenVMS system that require support for an IBM SNA logical unit (LU) type 2 session.

Note

> Unless otherwise stated, the term SNA Gateway or the 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. If, when using the OpenVMS SNA software, you recieve an error message that refers to the SNA Gateways or the Gateway, assume the message also refers to the OpenVMS SNA software.

Manual Objectives

The *Digital SNA 3270 Data Stream Programming Interface Guide* provides the information you need to write an application on an OpenVMS system to conduct an LU-LU type 2 session with a program residing in an IBM host.

Intended Audience

This manual is designed for OpenVMS programmers. To use the 3270 Data Stream Interface you need a general understanding of IBM's Systems Network Architecture (SNA), but you do not need a detailed knowledge of SNA.

Changes and New Features

The Digital SNA 3270 Data Stream Programming Interface for OpenVMS, Version 1.5 differs from the Version 1.4 product only in that it includes support for utilizing TCP/IP to exchange messages with a cooperating application on the IBM host.

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; and the default transport order for OpenVMS VAX will be local, decnet, tcp.

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

The *Digital SNA 3270 Data Stream Programming Interface Guide* is divided into four parts.

Part I
Contains Chapter 1, which provides a philosophical statement that addresses the questions "How do I know which DECnet SNA application interface product to buy?" and "What can I do with this product?".

Part II
Provides a tutorial that discusses use of the Interface and describes its features. It also contains a chapter of programming examples you can use as a guide to writing applications. Part II contains the following four chapters:

Chapter 2
Provides an overview of the Interface, describing its two modes of operation and, in general terms, how your OpenVMS application can make calls to it.

Chapter 3
Describes features of the Interface that help you write and execute your application.

Chapter 4
Describes the procedure for linking an OpenVMS application to the Interface using a shareable image.

Chapter 5
Provides programming examples in several commonly used languages to illustrate how your application makes calls to the Interface.

Part III
Provides reference material you need for writing an application that uses the Interface. It contains the following chapter:

Chapter 6
Presents the calling format and parameter list for each procedure provided by the Interface.

Part IV
Appendixes

Appendix A
Provides attention identification (AID) key values with symbols and keyboard equivalents.

Appendix B
Provides a summary of the notation used to describe parameters in the Interface.

Appendix C
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 D
Describes the status codes that the Interface returns to the OpenVMS application.

Appendix E
Provides low-level status codes that you might receive when you use the Interface.

Appendix F
Correlates procedures and status messages used by the Interface.

Associated Documents

The following is a list of documents related to the Digital SNA 3270 Data Stream Programming Interface:

- *Digital SNA 3270 Data Stream Programming Interface for OpenVMS Installation*
- *Digital SNA 3270 Data Stream Programming Interface for OpenVMS Problem Solving*
- *Digital SNA 3270 Data Stream Programming Interface for OpenVMS Programming*

You should have the following Digital documents available for reference when you use the Digital SNA 3270 Data Stream 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*

See the following documents for more information about the IBM 3270 Information Display System:

- *ACF for VTAM Version 2, Messages and Codes* (IBM Order No. SC27-0614)
- *IBM 3270 Information Display System and 3274 Control Unit Description and Programmer's Guide* (IBM Order No. GA23-0061)
- *IBM 3287 Printer Models 1 and 2 Component Description* (IBM Order No. GA27-3153)
- *MVS/TSO/VTAM Data Set Print Program Description/Operations Manual* (IBM Order No. SB21-2070)
- *IBM 3270 Information Display System*, Order No. GA23-0060
- *IBM 3270 Information Display System Data Stream Programmer's Reference*, Order No. GA23-0059
- *Systems Network Architecture—Introduction to Sessions Between Logical Units*, Order No. GC20-1869
- *Systems Network Architecture—Sessions Between Logical Units*, Order No. GC20-1868
- *IBM 3270 Information Display System: Operator's Guide*, Order No. GA27-2742

Conventions

This manual uses the following conventions:

Convention	Meaning
CAPITAL LETTERS	Represent constant values, or symbols. Code these exactly as they are specified.
lowercase italics	Represent variables for which you must supply a value.

Convention	Meaning
[ ]	Square brackets enclose parameters or symbols that are either optional or conditional. Specify the parameter and value if you want the condition to apply. Do not type the brackets in the line of code. The following rules generally apply to parameters: • You may code or omit an optional parameter. Omitting an optional parameter may impact a related parameter or may cause a default value to be specified. • You may code or omit a conditional parameter. Your choice is determined by how other parameters are coded.
( )	Parentheses delimit the argument list. The arguments must be typed in the line of code in the order indicated. Parentheses must be typed where they appear in a line of code.
`Special type`	Examples of system output and user input are printed in this special type.
Numbers	Numbers are decimal unless otherwise noted.
RET	Unless otherwise specified, every command line is terminated by pressing the RETURN key.
CTRL/*x*	Control characters are shown as CTRL/*x*, where *x* is an alphabetic character. The CTRL key and the appropriate key should be pressed simultaneously.

Terminology

When this manual refers to the OpenVMS application, it means the application the user writes. When the manual refers to the Application Interface, it means the Digital software that performs LU2 functions for the OpenVMS application.

This manual uses the following two abbreviations for Digital SNA 3270 Data Stream Programming Interface:

- the Interface
- the 3270 DS Interface

The term SNA Gateway or Gateway refers to any of the following Digital products:

- DECnet SNA Gateway-ST

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

Part I

Introduction

1 Introduction

The Digital SNA 3270 Data Stream Programming Interface allows you to develop OpenVMS applications that exchange messages with cooperating applications on an IBM host. In order to exchange these messages, the OpenVMS application requires support from the Interface to establish a session with an IBM SNA logical unit (LU) type 2. An LU type 2 session is a connection between two systems that exchange messages via the IBM 3270 data stream. This interface interacts with Digital's SNA Gateway to provide support for the LU type 2 session.

The 3270 DS Interface is one of three Digital SNA programming interface products. The other two are the Digital SNA Application Programming Interface and the Digital SNA APPC/LU6.2 Programming Interface. Both of these products also enable you to exchange messages with cooperating applications on an IBM host.

- The Digital SNA Application Programming Interface provides support to OpenVMS applications that need to establish a session with any LU (except LU type 6.2).
- 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.

To use the 3270 Data Stream Programming Interface, **your IBM system must support SNA LU type 2 sessions**. If your system does not support LU type 2 sessions, one of the other programming interfaces discussed above may be able to solve your programming needs.

1.1 3270 Data Stream Features

Using the 3270 Interface, a OpenVMS application can perform the following functions:

- Create a virtual 3270 display
- Receive an uninterpreted IBM 3270 data stream
- Access data on both your OpenVMS and IBM computer systems
- Connect to 3270 transactions already residing on your IBM host
- Perform inquiry/response transactions

Note

Do not use the Digital SNA 3270 Data Stream Programming Interface to update a distributed database. Because the Interface is unable to restart a session after a failure at exactly the point where the session failed, you cannot reliably update a database. If updating a distributed database is important to you, use the Digital SNA Application Programming Interface, which supports LU0.

The 3270 DS Interface has two modes of operation that you can use, depending upon the function you want to perform: data stream mode and field mode.

In **data stream mode**, the Interface establishes LU-LU type 2 sessions to send and receive uninterpreted 3270 data streams. In this mode, the Interface performs SNA services up to and including the Data Flow Control layer. It is the OpenVMS user's responsibility to:

- Interpret the data stream for orders and commands. For information about the data stream, see the *IBM 3270 Information Display System 3274 Control Unit Description and Programmer's Guide*, IBM Order No. GA23-0061-1.
- Build and manipulate a screen image (if desired).
- Build the data stream from the screen image before transmitting it to IBM.

In **field mode**, the Interface converts the received data stream into a virtual screen image. In this mode, the Interface performs SNA services up to and including the Presentation Services layer. The application can then update the screen by calling on the Interface to read and write specified fields. To return an updated screen to the IBM host, the Interface converts the screen image back into a 3270 data stream for transmission.

1.2 SNA Concepts

You can develop a variety of applications using the 3270 Data Stream Programming Interface. Because the Interface sends, receives, and interprets SNA protocol messages for you, you need not be concerned with SNA message formats and protocols. Nevertheless, this manual assumes a general knowledge of SNA. Familiarity with the following concepts will help you use the 3270 Data Stream Programming Interface efficiently:

- **General nature of IBM's LU type 2**—The LU type 2 provides a 3270 type display and uses a 3270 data stream to transmit data.
- **Half-duplex flip-flop communications**—Two LUs in communication with each other use this mechanism to alternate sending data to one another.
- **Session processing states**—These states control the processing of SNA commands, responses, and user data transmissions.
- **Bracketing**—This is the way SNA groups data into logical entities (complete transactions).
- **Chaining**—This is the way SNA breaks large blocks of data into pieces for transmission.

For further details and complete descriptions of these concepts, please see the *IBM 3270 Information Display System 3274 Control Unit Description and Programmer's Guide*, IBM Order No. GA23-0061-1, and *Systems Network Architecture–Sessions Between Logical Units*, IBM Order No. GC20-1868.

1.3 Common Interface Applications

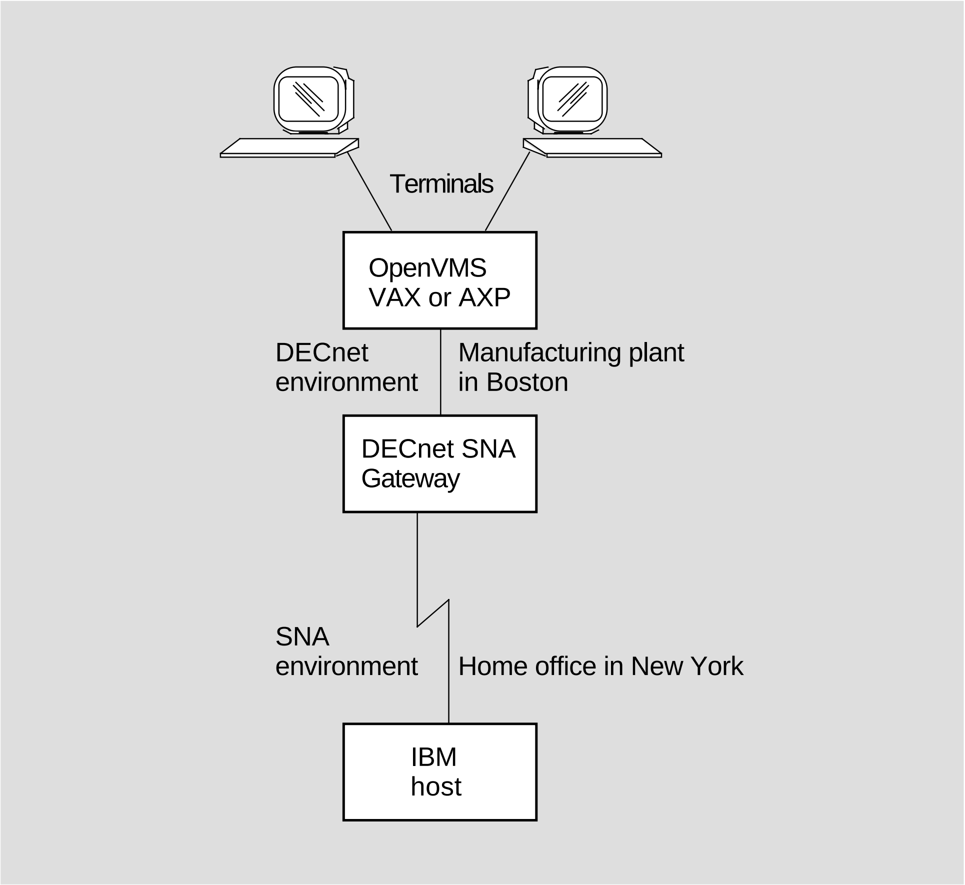
You can use the Digital SNA 3270 Data Stream Programming Interface to write applications that make up the secondary logical unit (SLU) half-session partner in an LU type 2 session. For example, you can write:

- An application that emulates functions of an IBM 3270 terminal.
- A file-transfer application that uses the 3270 data stream to communicate with a third-party software product running in the IBM host.
- An OpenVMS application that needs to communicate with an existing IBM application that was designed for 3270 terminal input/output.

For instance, the United States Widget Manufacturing Company wishes to collect inventory information using OpenVMS systems. Widget uses an OpenVMS system in its manufacturing plant to track local parts inventory while storing companywide inventory in the IBM data center at company headquarters.

To keep the companywide inventory current, the OpenVMS system must transmit an updated local inventory to the IBM host once a day after the close of business. Using the 3270 Data Stream Programming Interface and going through a SNA Gateway, the OpenVMS application can connect to a cooperating transaction on the IBM host and transmit the day's transactions. Figure 1–1 shows Digital terminals linked to a an SNA Gateway in the DECnet environment. Note that this DECnet environment could also be TCP/IP.

Figure 1–1 DECnet SNA Network



LKG−8074−93R

Part II

Tutorial

2

3270 Data Stream Programming Interface Overview

The Digital SNA 3270 Data Stream Programming Interface consists of procedures that a user-written OpenVMS program can call to request the following operations for an LU-LU type 2 session:

- Establish a session with an IBM application
- Receive and transmit a 3270 data stream
- Receive and transmit a virtual 3270 screen image
- Read and write fields in a 3270 screen image

2.1 Establishing an LU-LU Type 2 Session

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

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 desired LU. 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 to the PLU a control initiate (CINIT) request. The PLU in turn sends a BIND request to the SLU, proposing the conditions of the session. The SLU examines the BIND request and accepts or rejects the session.

In type 2 sessions involving the SNA Gateway, the Interface performs LU functions for the user application. In these sessions the Interface is always the SLU. This means that the Interface is always the receiver, never the sender, of the BIND.

The OpenVMS application can issue two kinds of requests to the Interface to establish an LU type 2 session: an active connect request and a passive connect request.

- An active connect request informs the Interface that the OpenVMS application wants to send an INIT-SELF to the SSCP to initiate a session with a specified IBM application.
- A passive connect request informs the Interface that the OpenVMS application is ready to engage in a session initiated by an IBM application.

To issue an active or passive connect request, the OpenVMS application calls the SNA3270$REQUEST_CONNECT procedure. Input parameters include the following information that the application passes to the Interface:

- **An active/passive connect indicator**

  A value indicating whether this is an active or passive connect request.

- **A mode type indicator**

  A value indicating whether this is a data stream mode connection or a field mode connection. Data stream mode is described in Section 2.3; field mode is described in Section 2.4.

- **A Gateway DECnet node name or TCP/IP host name**

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

- **IBM access information**

  An identifier 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 3.4.

Output parameters for the SNA3270$REQUEST_CONNECT procedure include locations to receive the following information from the IBM system:

- **A session identifier**

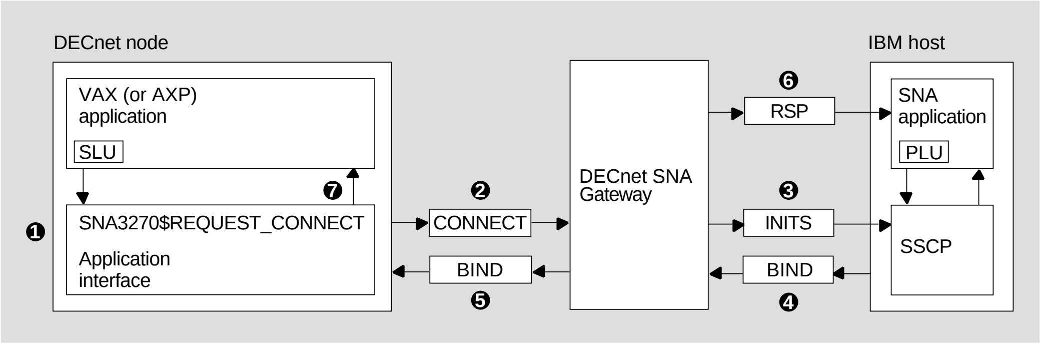
  A location to receive a unique identifier assigned by the Interface to the session. Each time the application issues a request to send or receive a data stream on this session, send a signal request, or terminate the session, the parameter list for the call must include the session identifier.

For a complete list of parameters for the SNA3270$REQUEST_CONNECT procedure, see Section 6.6.

In a typical active request for a session, the following steps occur (see Figure 2–1).

1. The OpenVMS application calls the SNA3270$REQUEST_CONNECT procedure, setting the active request indicator and providing the other required parameters.
2. The Interface sends a connect request to the SNA Gateway.
3. The Gateway sends an INIT-SELF request to the SSCP. The SSCP notifies the PLU.
4. The PLU sends a BIND request to the Gateway.
5. The Gateway sends the BIND request to the Interface. The Interface examines the BIND, accepts the session, and sends a response to the Gateway.
6. The Gateway forwards the positive response to the PLU.
7. The Interface completes the connect request by returning control to the user, setting an event flag, or calling an asynchronous system trap (AST) routine.

Figure 2–1 An Active Connect Request



LKG−8075−93R

A typical passive request for a session includes the following steps.

1. The OpenVMS application calls the SNA3270$REQUEST_CONNECT procedure, setting the passive request indicator and providing the other required parameters.
2. The Interface sends a message to the SNA Gateway indicating that the OpenVMS application is ready to receive a BIND request from the IBM application.
3. At some point, the PLU sends a BIND request to the Gateway.
4. The Gateway sends the BIND request to the Interface. The Interface examines the BIND request, accepts the session, and notifies the Gateway.
5. The Gateway sends a positive response to the PLU.
6. The Interface completes the connect request by returning control to the OpenVMS application, setting an event flag, or calling an AST routine.

2.2 Specifying a Connection Mode

The 3270 Data Stream Programming Interface provides two modes of connection for sessions established between an OpenVMS application and an IBM application subsystem: data stream mode and field mode.

- In data stream mode, the OpenVMS application and the application subsystem exchange complete 3270 data streams as defined by IBM. Data stream mode is described in Section 2.3.
- In field mode, the OpenVMS application and the application subsystem exchange data in the form of virtual screen images consisting of various data fields. Field mode is described in Section 2.4.

2.3 Data Stream Mode

As defined by IBM, a complete 3270 data stream is a sequence of application data, commands, structured field functions, orders, and control information that normally forms a complete screen buffer. The complete stream is transmitted between a cluster controller and a host. For details on the 3270 data stream, see the *IBM 3270 Information Display System: 3274 Control Unit Description and Programmer's Guide.*

To establish a session for the purpose of exchanging a 3270 data stream with an IBM host, the OpenVMS application calls the SNA3270$REQUEST_CONNECT procedure and sets the connection mode indicator in the parameter list to specify data stream mode.

Once the session is established, the OpenVMS application can call procedures provided by the Interface to transmit and receive 3270 data streams.

2.3.1 Transmitting a 3270 Data Stream

To transmit a 3270 data stream, the OpenVMS application calls the SNA3270$TRANSMIT_STREAM procedure, indicates a session, and specifies a buffer that contains a complete or partial data stream. The OpenVMS application must leave room in the buffer for SNA header information. The number of bytes that you must reserve is defined by the symbol SNA3270$K_BUF_HDLEN.

- If the OpenVMS application indicates that the buffer contains a complete data stream (see the *last-flag* parameter in Section 6.11), the Interface transmits the data and relinquishes the OpenVMS application's turn to send.
- If the application indicates a partial data stream, the Interface transmits the data but does not relinquish the OpenVMS application's turn to send.

When the procedure has completed, the buffer is immediately available for reuse.

In normal LU type 2 communications, the OpenVMS application and the IBM application subsystem alternate between sending and receiving data streams. In certain cases, however, the OpenVMS application may wish to send two or more consecutive streams. If the OpenVMS application has relinquished its turn to send, the application can request to send another stream in one of two ways:

- If the last transmit completed with the status SNA3270$_OK_CONT, the OpenVMS application can send another stream.
- If the last transmit completed with the status SNA3270$_OK, it is the IBM application's turn to send. The OpenVMS application can send a SIGNAL request asking the IBM application to relinquish its turn to send.

In the second case, the IBM application is not obligated to honor the SIGNAL request. Therefore, the Interface does not wait to receive the flag indicating who can send before returning from the SNA3270$TRANSMIT_SIGNAL procedure call. Thus, the OpenVMS application does not know whether the IBM application accepted the SIGNAL request or whether it is giving up its turn to send. The application will always get an error if it tries to send when it is not its turn. After sending the SIGNAL request the application should post a receive, which eventually completes with SNA3270$_OK_CONT or SNA3270$_OK.

2.3.2 Receiving a 3270 Data Stream

To receive a 3270 data stream, the OpenVMS application calls the SNA3270$RECEIVE_STREAM procedure, indicates a session, and specifies a buffer to contain the data stream.

In the simplest case, the Interface fills the buffer and returns a status message to indicate that the buffer contains a complete 3270 data stream. The first SNA3270$K_BUF_HDLEN bytes contain SNA header information, which you can ignore. If the buffer is too small to contain a complete stream, the Interface returns a status message to indicate that the procedure has completed successfully but that more data remains. The OpenVMS application then issues additional calls to the SNA3270$RECEIVE_STREAM procedure until the Interface returns a status message to indicate that the complete stream has been transferred.

If a receive completes with the status SNA3270$_OK_MORE (more data) or SNA3270$_OK_NYT (not your turn) issue another receive to receive more data or to wait for IBM to relinquish its turn to send.

2.3.2.1 Acknowledging the Stream

Once the complete stream has been transferred, the OpenVMS application must inform the IBM application subsystem that the received data is acceptable or unacceptable. To do this, the OpenVMS application calls the SNA3270$ACKNOWLEDGE procedure and specifies the appropriate positive or negative response. If you call the SNA3270$TRANSMIT_STREAM procedure before calling the SNA3270$ACKNOWLEDGE procedure, the Interface performs an implied positive acknowledge. In other words, the acknowledge is only required to acknowledge the received stream negatively.

2.3.2.2 Receiving a Request to Send Multiple Consecutive Streams

In normal LU type 2 communications, the OpenVMS application and the IBM application subsystem alternate between sending and receiving data streams.

At any point, however, the IBM subsystem may issue a request to the Interface to send another data stream immediately after the one just sent. If the IBM application relinquished its turn to send, it can request to send another stream in one of two ways:

- If the last receive or transmit completed with the status SNA3270$_OK_CONT, the IBM application can BID to send another stream.

  The Interface accepts the BID request and notifies the OpenVMS application when it has received the data. The OpenVMS application can cause the Interface to reject the BID by calling the SNA3270$LOCK_SCREEN procedure before the BID request is received. When using

SNA3270$LOCK_SCREEN, the OpenVMS application is saying it wants to be sure the PLU will not send more data while the OpenVMS application is preparing to send data. Once a BID request has been accepted, a call to the SNA3270$LOCK_SCREEN procedure will have no effect.

- If the last receive completed with the status SNA3270$_OK, it is the OpenVMS application's turn to send. IBM can send a SIGNAL request asking the OpenVMS application to relinquish its turn to send.

The Interface is obligated to accept the SIGNAL request and relinquish its turn to send to the IBM application. The Interface notifies the OpenVMS application that it has lost its turn to send. In this case, the OpenVMS application cannot call the SNA3270$LOCK_SCREEN procedure and cause the Interface to reject the SIGNAL request.

2.4 Field Mode

In field mode, the OpenVMS application receives and sends information by means of two display vectors—a character vector and an attributes vector—that represent a 3270 block-mode screen image consisting of one or more fields. Because these two vectors contain all the information necessary to produce a 3270 block-mode screen image on a Digital terminal, they are said to represent a virtual 3270 screen image.

To establish a session for the purpose of receiving and sending virtual 3270 screen images, the OpenVMS application calls the SNA3270$REQUEST_CONNECT procedure, sets the connection mode indicator in the parameter list to specify field mode, and provides a character vector and an attribute vector to contain the virtual 3270 screen images. As part of the request procedure, the Interface builds a structure called a screen descriptor block, which it uses to supply information about the vectors.

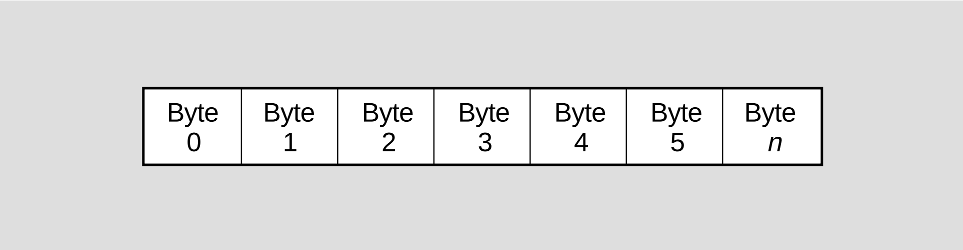
Once the session has been established, the OpenVMS application typically issues calls to request the following sequence of operations:

1. Receive a 3270 screen image from the application subsystem in the IBM host
2. Read fields in the screen image
3. Write fields in the screen image
4. Transmit the screen image to the IBM host

2.4.1 The Character Vector

The character vector contains a series of 8-bit EBCDIC characters that correspond to the characters displayed in a screen image. Figure 2–2 illustrates a character vector. Each character is assigned a vector address that corresponds to a screen position.

Figure 2–2 Character Vector



LKG−8076−93R

Note

A screen position is normally described by row and column coordinates *x* and *y* starting from position 1. To translate these coordinates into a vector address, the application can use the formula

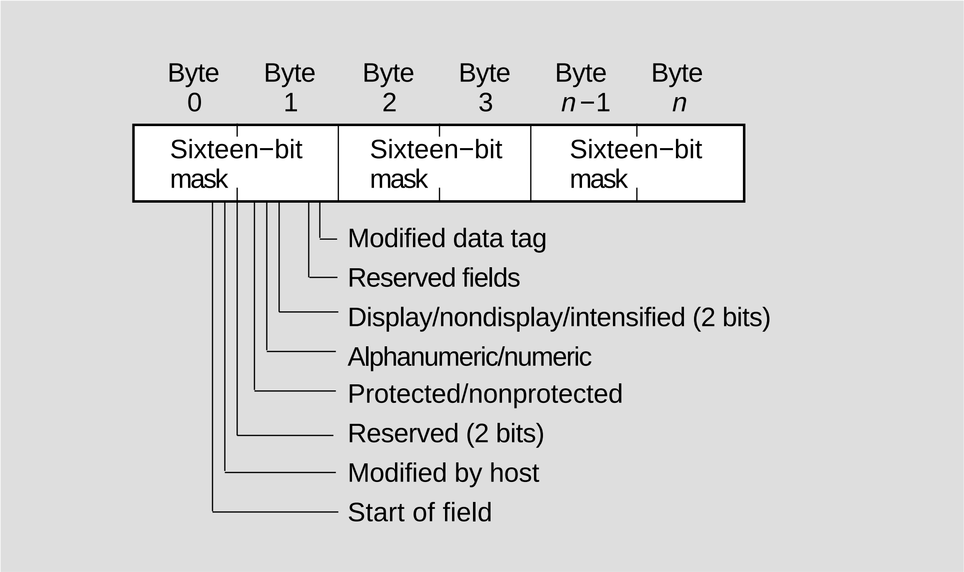
$$i = c(x - 1) + (y - 1)$$

where *i* is a vector position and *c* is the number of columns in the display. For example, in an 80-column display, the character in row 5, column 22 is in vector position 80(4) + 21, or 341.

2.4.2 The Attributes Vector

The attributes vector is built by the Interface using the field attribute character(s) in the 3270 data stream. The attributes vector consists of a series of 16-bit words. Each word is a mask that specifies the attributes of the corresponding character in the character vector. For example, the mask specifies whether the character is located at the beginning of a field, whether it is protected or nonprotected, or whether it has been modified by the IBM host or the OpenVMS application. Figure 2–3 illustrates the attributes vector.

Figure 2–3 Attributes Vector With Mask Flags



LKG−8077−93R

Table 2–1 lists the symbols and meanings of the attributes masks in order from least significant to most significant bit.

Table 2–1 Symbols and Meanings of Attribute Masks

Symbol	OpenVMS Position (Bit)	Meaning
Masks Used With the Attributes Vector		
SNA3270$M_ATTR_MDT	0	Modified data tag
SNA3270$M_ATTR_DSP	2,3	Display/Intensity
SNA3270$M_ATTR_NUM	4	Numeric
SNA3270$M_ATTR_PRO	5	Protected
SNA3270$M_ATTR_MBH	8	Modified by host

(continued on next page)

Table 2–1 (Cont.) Symbols and Meanings of Attribute Masks

Symbol	OpenVMS Position (Bit)	Meaning
Masks Used With the Attributes Vector		
SNA3270$M_ATTR_SOF	9[1]	Start of field
The SNA3270$M_ATTR_DSP has four possible values:		
SNA3270$K_ATTR_NORM		Normal intensity, nondetectable
SNA3270$K_ATTR_PEN_DET		Normal intensity, detectable
SNA3270$K_ATTR_HIGH		High intensity, detectable
SNA3270$K_ATTR_INVIS		Nondisplayed, nondetectable

[1] The start-of-field bit indicates whether the corresponding character in the character vector is at the start of a field.

Note

The display/intensity values listed in Table 2–1 are values to be inserted into a 2-bit field (the values are 0, 1, 2, and 3, respectively). Many languages do not have the ability to insert values into a bit field; therefore, you must shift the values 0, 1, 2, or 3 left by two places. Then use an inclusive OR to insert the values into the field.

2.4.3 The Screen Descriptor Block

For each field mode session requested, the OpenVMS application allocates a data structure called the screen descriptor block (SDB). The Interface uses this block to provide the application with information about the 3270 screen image in the display vectors. Figure 2–4 illustrates an SDB. A brief description of the components of the SDB follows:

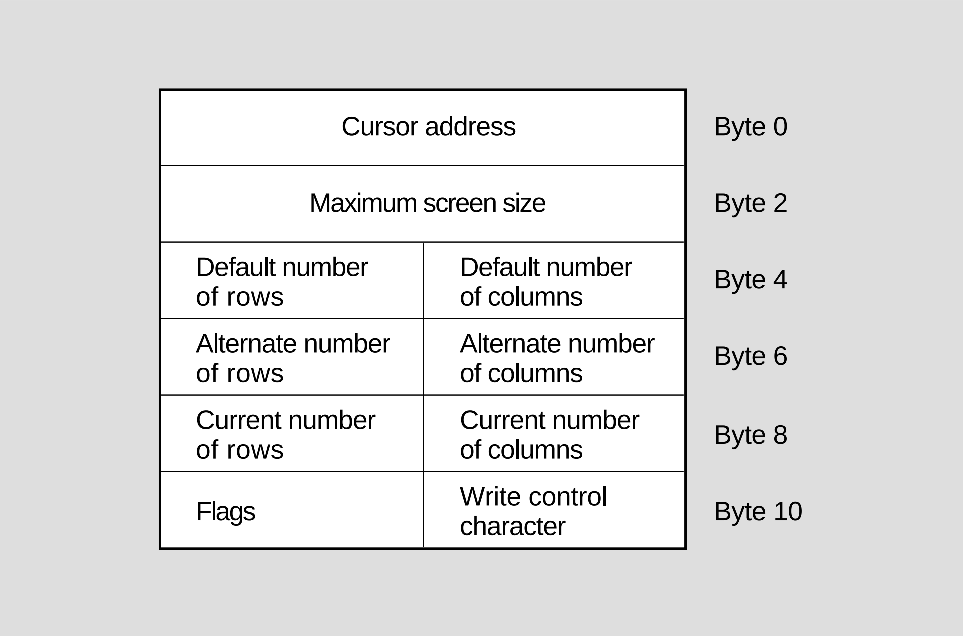
- Current cursor address: If the OpenVMS application uses the SNA3270$READ_FIELD and SNA3270$WRITE_FIELD procedures, the Interface updates this address automatically. If the OpenVMS application modifies the display vectors directly, the application must also update the cursor address. Initially, the IBM system sets the value of this field with an insert cursor order in the data stream.
- Current screen size: The total number of characters in the display.

- Default number of rows and columns specified at BIND time.
- Alternate number of rows and columns specified at BIND time.
- The current number or rows and columns in the display.
- Flags that describe the action that affected the screen image. A list of these flags and their meanings follows:

Symbol	Meaning
SNA3270$M_SDB_RESET_INHIB	Reset keyboard inhibit condition
SNA3270$M_SDB_PSA_PEND	Presentation space is altered
SNA3270$M_SDB_SCR_ERASED	Screen is erased
SNA3270$M_SDB_SIZE_CHANGED	Screen size is changed
SNA3270$M_SDB_FORMAT_SCR	Screen is formatted
SNA3270$M_SDB_WRAP_SCR	Screen is wrapped

- Write control character from the 3270 data stream. This value should not be modified by the application.

Figure 2–4 The Screen Descriptor Block



LKG−8078−93R

Table 2–2 lists the field names and associated symbols for the screen descriptor block. All of the fields are used for output.

Table 2–2 The Screen Descriptor Block and Associated Symbols

Field Name	Position (Byte)	Symbol
Cursor address	0,1	SNA3270$W_SDB_CURSOR_ADDR
Maximum screen size	2,3	SNA3270$W_SDB_MAX_SCRSIZE
Default number of columns	4	SNA3270$B_SDB_DEF_COLS
Default number of rows	5	SNA3270$B_SDB_DEF_ROWS

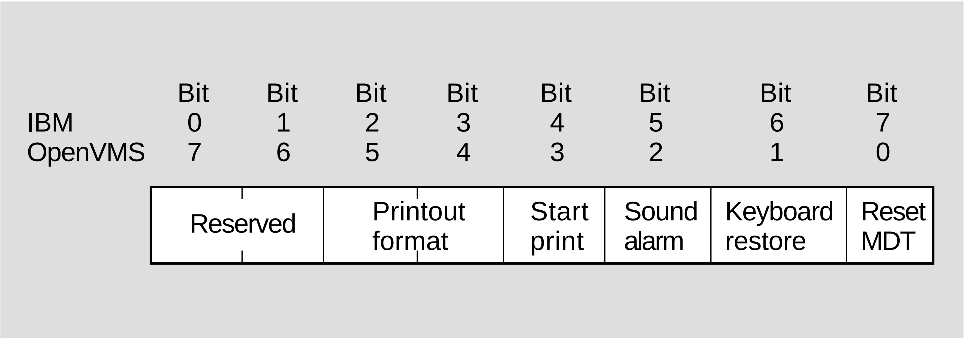
(continued on next page)

Table 2–2 (Cont.) The Screen Descriptor Block and Associated Symbols

Field Name	Position (Byte)	Symbol
Alternate number of columns	6	SNA3270$B_SDB_ALT_COLS
Alternate number of rows	7	SNA3270$B_SDB_ALT_ROWS
Current number of columns	8	SNA3270$B_SDB_CUR_COLS
Current number of rows	9	SNA3270$B_SDB_CUR_ROWS
Write control character	10	SNA3270$B_SDB_WCC
Flags	11	SNA3270$B_SDB_FLAGS

Figure 2–5 illustrates the write control character and the symbols and meanings that represent the various bits. It also illustrates the different way IBM and OpenVMS systems number bits for the write control character.

Figure 2–5 Write Control Character



LKG−8079−93R

Table 2–3 lists the names and associated symbols for the write control character.

Table 2–3 Write Control Character and Associated Symbols

Name	OpenVMS Position (Bit)	Symbol
Reset modified data tags	0	SNA3270$M_WCC_RESET_MDT
Keyboard restore	1	SNA3270$M_WCC_KBD_RST
Sound audible alarm	2	SNA3270$M_WCC_ALARM
Start print	3	SNA3270$M_WCC_PRINT
Printer format (not used under SNA)	4,5	SNA3270$M_WCC_PRT_FMT

2.4.4 Receiving a 3270 Screen Image

To receive a virtual 3270 screen image from the IBM host, the OpenVMS application calls the SNA3270$RECEIVE_SCREEN procedure and indicates the session. The Interface places a complete virtual screen image in the display vectors specified for the session.

While the display vectors are being modified, the user is prevented from altering information in the vectors. Once the Interface has finished modifying the screen image in the display vectors, the OpenVMS application can begin reading and modifying fields in the display vectors.

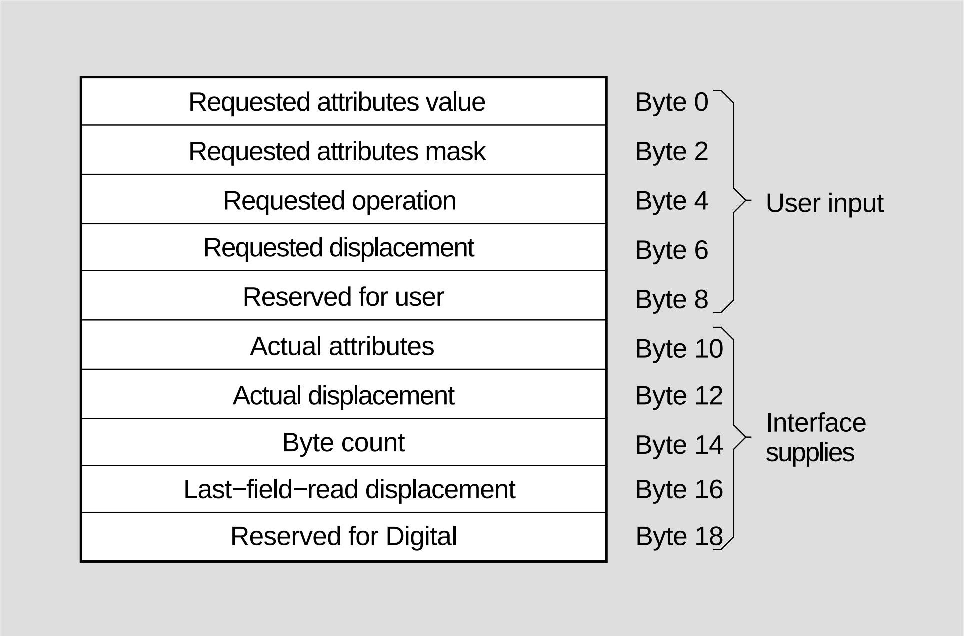
Note

The SNA3270$READ_FIELD and SNA3270$WRITE_FIELD procedures do not allow vector modification if the keyboard restore bit is set off. The OpenVMS application is responsible for this function if it does not use SNA3270$READ_FIELD and SNA3270$WRITE_FIELD. For instance, data may continue coming in and you will modify an incomplete screen.

2.4.5 Retrieving Fields From a 3270 Screen Image

To retrieve a field from a 3270 virtual screen image, the OpenVMS application calls the SNA3270$READ_FIELD procedure with the field descriptor block (FDB). The FDB is specified in the SNA3270$REQUEST_CONNECT procedure and is used throughout the session. The FDB provides space for the offset address of the target field, attributes that the Interface should use to locate the field, and symbols to specify the desired operation. Figure 2–6 illustrates the FDB. The OpenVMS application also specifies a buffer to receive the read field.

Figure 2–6 The Field Descriptor Block



LKG−8080−93R

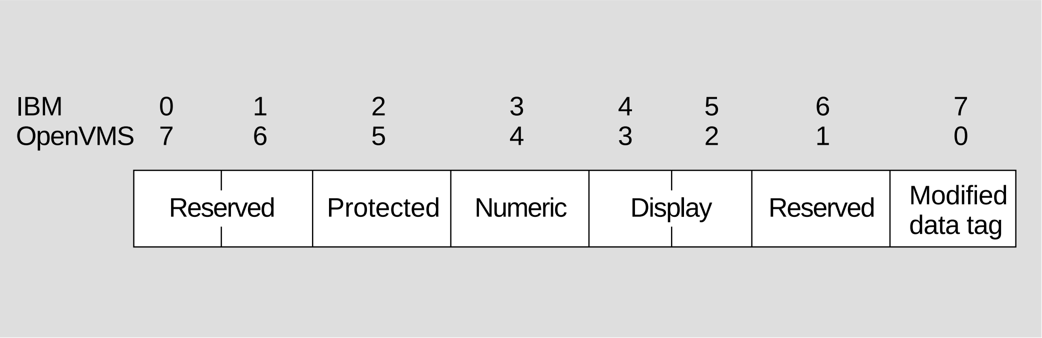
Table 2–4 lists the field names and associated symbols for the field descriptor block. The table also indicates whether the field is used for input or for output.

Table 2–4 Field Descriptor Block Fields and Associated Symbols

Field Name	Position (Byte)	Symbol	Input/Output
Requested attributes value	0,1	SNA3270$W_FDB_ATT_VALUE	Input to READ_FIELD
Requested attributes mask	2,3	SNA3270$W_FDB_ATT_MASK	Input to READ_FIELD
Requested operation	4,5	SNA3270$W_FDB_SELECT	Input to READ_FIELD
Requested displacement	6,7	SNA3270$W_FDB_BUFOFF	Input to READ_FIELD
Reserved for customer	8,9	SNA3270$W_FDB_FILLER2	
Actual attributes	10,11	SNA3270$W_FDB_FLD_ATTR	Output from READ_FIELD, WRITE_FIELD
Actual displacement	12,13	SNA3270$W_FDB_FLD_ BUFOFF	Output from READ_FIELD Input to WRITE_FIELD
Byte count	14,15	SNA3270$W_FDB_FLD_COUNT	Output from READ_FIELD
Last-field-read displacement	16,17	SNA3270$W_FDB_LFR_ BUFOFF	Output from READ_FIELD

Figure 2–7 illustrates the symbols and meanings for the various bits in the requested attributes value and requested attributes mask fields of the FDB.

Figure 2–7 Attributes and Associated Symbols for the Field Descriptor Block



LKG−8081−93R

The following list provides a description of the fields in the FDB:

- **Requested attributes value:** This field works in conjunction with the requested attributes mask field. Specify the attribute values of the field which the application needs to locate and read, using the symbols defined in Table 2–5. For example, to locate a numeric-only field the OpenVMS application would set this field to SNA3270$M_ATTR_NUM.
- **Requested attributes mask:** This field works in conjunction with the requested attributes value field. This field is used to specify which attributes the application is interested in checking. A bit set in this field causes the Interface to compare the corresponding attributes with the value specified in the requested attributes value field. A bit clear in this field causes the Interface to ignore the corresponding attribute. For example, to specify a nonprotected, numeric field in FORTRAN with the FDB defined with the statement RECORD /SNA3270_FDB/ FDB, the requested value and attribute fields would be set as follows:

  FDB.SNA3270$W_FDB_ATT_MASK = SNA3270$M_ATTR_NUM + SNA3270$M_ATTR_PRO

  FDB.SNA3270$W_FDB_ATT_VALUE = SNA3270$M_ATTR_NUM

  In this example, the mask specifies that the user is interested in only the numeric and protected attributes. The value specifies that the numeric attribute flag should be set and the protected attribute flag should be reset.

Table 2–5 lists the names and associated symbols for the requested attributes in the FDB.

Table 2–5 Field Descriptor Block Attributes and Associated Symbols

Name	OpenVMS Position (Bit)	Symbol
Modified data tag	0	SNA3270$M_ATTR_MDT
Displayable/intensity /light pen detectable	2,3	SNA3270$M_ATTR_DSP
Numeric	4	SNA3270$M_ATTR_NUM
Protected	5	SNA3270$M_ATTR_PRO
Modified by host	8	SNA3270$M_ATTR_MBH
Start of field	9	SNA3270$M_ATTR_SOF

- **Requested operation:** This field indicates the operation the application desires to perform.

Table 2–6 provides symbols and meanings you can specify for the attributes and operation selection fields of the FDB.

Table 2–6 Symbols and Meanings of the Operation Selection Fields of the Field Descriptor Block

Symbol	Meaning
SNA3270$K_SEL_READ	Read at specified offset
SNA3270$K_SEL_SEARCH	Search from specified offset
SNA3270$K_SEL_READ_NEXT	Read next field
SNA3270$K_SEL_SEARCH_NEXT	Search starting from next field

- **Requested displacement:** The offset at which you want to begin searching or reading a field.
- **Reserved for user:** A field reserved for the user to include such information as a pointer or a counter.
- **Actual attributes:** The actual attributes of the field you just read or are about to write into.
- **Actual displacement:** The actual offset of the field you searched for and read is returned in this location.

- **Byte count:** The actual number of bytes read from or written to a field.
- **Last-field-read displacement:** The offset of the last field read by the Interface. This field is used in the search operation for the SNA3270$K_SEL_READ_NEXT or the SNA3270$K_SEL_SEARCH_NEXT operation.
- **Reserved for Digital:** Empty fields reserved for future releases of the product.

The OpenVMS application can use the FDB to specify the following types of read and search operations to retrieve a field. The specified displacement should always be the address of the attributes of the field.

- **Read the field that begins at the specified offset address**

  The OpenVMS application provides an FDB containing an offset address with the requested operation field set to SNA3270$K_SEL_READ to specify the operation. If the offset is the beginning of a field, the Interface returns the field to the specified buffer. If the offset is not the beginning of a field, the Interface returns the SNA3270$_BADOFFSET status value.

- **Read the field that begins at the specified offset address if the field has the specified attribute**

  The OpenVMS application provides an FDB containing an offset address, an attribute mask and value, and the requested operation field set to SNA3270$K_SEL_READ to specify the operation. If the offset is the beginning of a field and the field has the specified attribute, the Interface returns the field. If the offset is not the beginning of a field, the Interface returns the SNA3270$_BADOFFSET status value; or if the field does not have the specified attribute, the Interface returns the SNA3270$_NOFIELD status value.

- **Read the next field**

  The OpenVMS application provides an FDB with the requested operation field set to SNA3270$K_SEL_READ_NEXT to specify the operation. The Interface returns the next field. If no field follows the last field read, the Interface returns the SNA3270$_NOFIELD status value.

- **Read the next field if the field has the specified attribute**

  The OpenVMS application provides an FDB containing an attribute mask and value with the requested operation field set to SNA3270$K_SEL_READ_NEXT to specify the operation. If the next field has the specified attribute, the Interface returns the field. If not, the Interface returns the SNA3270$_NOFIELD status value.

- **Search for a field, starting at the specified offset address**

  The OpenVMS application provides an FDB containing an offset address with the requested operation field set to SNA3270$K_SEL_SEARCH to specify the operation. Starting at this offset, the Interface searches the screen image sequentially until it encounters the beginning of a field. It returns this field to the OpenVMS application. If the specified offset is out of range, the Interface returns the SNA3270$_BADOFFSET status value. If no field begins after this offset, the Interface returns the SNA3270$_NOFIELD status value.

- **Search for a field with the specified attribute, starting at the specified offset**

  The OpenVMS application provides an FDB containing an offset address, an attribute mask and value, and the requested operation field set to SNA3270$K_SEL_SEARCH to specify the operation. Starting at the specified offset, the Interface searches the screen image sequentially for a field with the specified attribute. If it finds such a field, it returns it to the OpenVMS application. If the specified offset is out of range, the Interface returns the SNA3270$_BADOFFSET status value. If the Interface does not locate the field, it returns the SNA3270$_NOFIELD status value.

- **Search for a field, starting after the last field read**

  The OpenVMS application provides an FDB with the requested operation field set to SNA3270$K_SEL_SEARCH_NEXT to specify the operation. Starting after the last field read, the Interface searches sequentially for the next field. If the screen image contains another field after the last field read, the Interface returns it to the OpenVMS application. If not, the Interface returns the SNA3270$_NOFIELD status value.

- **Search for a field with the specified attribute, starting after the last field read**

  The OpenVMS application provides an FDB containing an attribute mask and value and the requested operation field set to SNA3270$K_SEL_SEARCH_NEXT to specify the operation. Starting after the last field read, the Interface searches for a field with the specified attribute. If the screen image contains such a field, the Interface returns it. If not, it returns the SNA3270$_NOFIELD status value.

2.4.6 Using the FDB: An Example

The following FORTRAN programming fragment will take you step by step through the use of the FDB. Interspersed between pieces of code you will find commentary and illustrations to explain what the program is doing. Ellipses represent information that has been omitted from the FORTRAN program.

The goal of the program is to show you how to extract fields from a typical IBM application screen by using the FDB. The program begins by establishing a session with IBM and providing an FDB. It then locates the first unprotected field. This sample OpenVMS program writes the AMNU IBM transaction. Upon completion, the entire screen of data is sent to IBM. The IBM responds with a screen of information from which the program can make a selection for the operation it desires. The program searches for the first unprotected field. It then writes the ABRW browse option. The program searches for the next unprotected field and the transaction number in that field. The IBM then responds with the browse screen.

At this point, the programming example presented here comes to an end. In an actual work environment, the application would continue reading and searching for fields in a manner similar to that used to locate fields in the AMNU transaction. For instance, the FDB can be used to read the employee number of John Doe and update his pay using the AUPD option.

```
        INCLUDE 'SYS$LIBRARY:SNA3270DF/NOLIST'  ! Include 3270
                                                ! definitions
```

A library of 3270 definitions has been loaded. The following lines define session variables.

```
        INTEGER*4 STATUS_VECTOR (SNA3270$K_MIN_STATUS_VECTOR)
        INTEGER*4 NOTIFY_VECTOR (SNA3270$K_MIN_NOTIFY_VECTOR)
        INTEGER*4 SESSION_ID
        INTEGER*4 RETURN_CODE, CONN_TYPE

                          .
                          .
                          .
```

The following lines define field mode structures. The FDB is a data block used to (1) describe fields in the screen image (Figure 2–4), and (2) supply the address and/or attributes of the field it will be reading in future SNA3270$READ_FIELD and SNA3270$WRITE_FIELD calls.

```
STRUCTURE /DSC/
        INTEGER*2 DSC$W_LENGTH
        BYTE DSC$B_DTYPE, DSC$B_CLASS
        INTEGER*4 DSC$A_POINTER
 END STRUCTURE
 RECORD /SNA3270_SDB/ SDB
 RECORD /SNA3270_FDB/ FDB
 RECORD /DSC/ SDB_DSC
 RECORD /DSC/ FDB_DSC
 INTEGER*4 SDB_DSC(2), FDB_DSC(2)
 CHARACTER*256 FIELD_DATA   INTEGER*2 FIELD_ATTR,
 1                          FIELD_OFFSET, LENGTH
```

.
.
.

In the next code, the SNA3270$REQUEST_CONNECT procedure establishes a field mode session between the OpenVMS application and the IBM application. The procedure points to the FDB in the event the Interface needs the FDB for future operations.

```
RETURN_CODE = SNA3270$REQUEST_CONNECT_W (SESSION_ID,
1 %DESCR(STATUS_VECTOR), %REF(SNA3270$K_ACTIVE),
2 %REF(SNA3270$K_FIELD_MODE), %DESCR(NODE_NAME),
3 %DESCR(ACC_NAME),,,,,,,, SCREEN_IMAGE,
4 %DESCR(ATTR_VECTOR),
5 %DESCR(FIELD_VECTOR), SDB_DSC, FDB_DSC, NOTIFY_RTN,
6 SESSION_ID, %DESCR(NOTIFY_VECTOR), %REF(LU2_EFN))

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

.
.
.

You will then receive the CICS logo screen. The program sends a clear screen request. IBM will respond with a cleared (blank) screen.

The following program code transmits AMNU to the IBM application. The code begins by describing the field to be located as an unprotected field.

```
FDB.SNA3270$W_FDB_ATT_MASK = SNA3270$M_ATTR_PRO
FDB.SNA3270$W_FDB_ATT_VALUE = 0
```

The program searches for the first unprotected field beginning from character position 0.

```
        FDB.SNA3270$W_FDB_BUFOFF = 0
        FDB.SNA3270$W_FDB_SELECT = SNA3270$K_SEL_SEARCH
```

The first unprotected field on the screen is located.

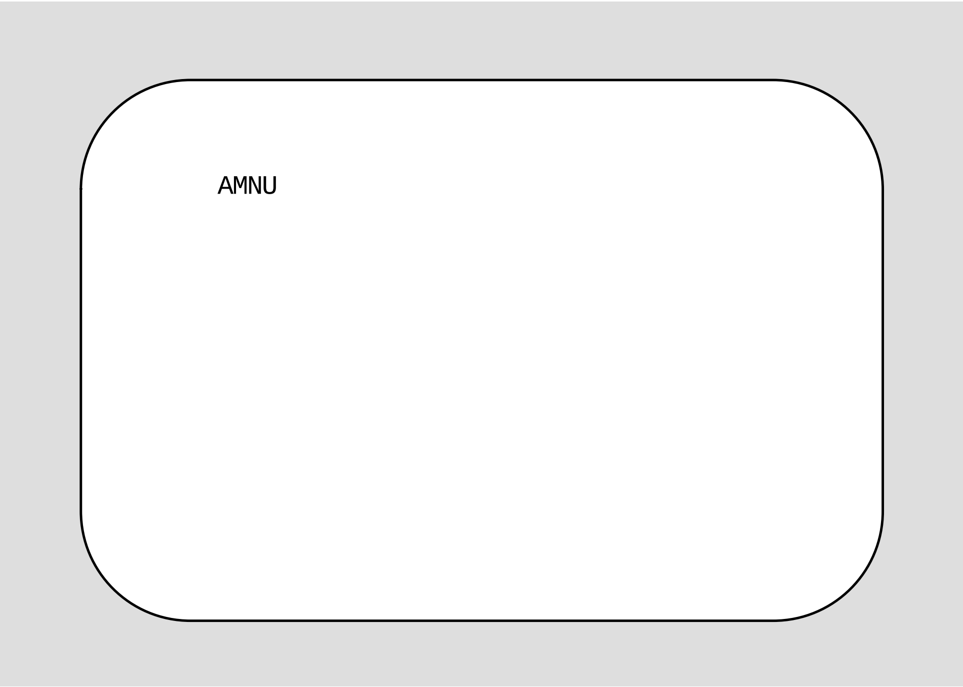
```
        RETURN_CODE = SNA3270$READ_FIELD (SESSION_ID,
        1 %DESCR(STATUS_VECTOR))

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

AMNU is set as the transaction name, and the string is converted to EBCDIC.

```
        FIELD_DATA = 'AMNU'
        ISTAT = LIB$TRA_ASC_EBC (DATA, DATA)
        IF (.NOT. ISTAT) CALL LIB$STOP(%VAL(ISTAT))
```

Using the SNA3270$WRITE_FIELD procedure, AMNU is entered into the field the OpenVMS application just located (see Figure 2–8).

Figure 2–8   AMNU Screen



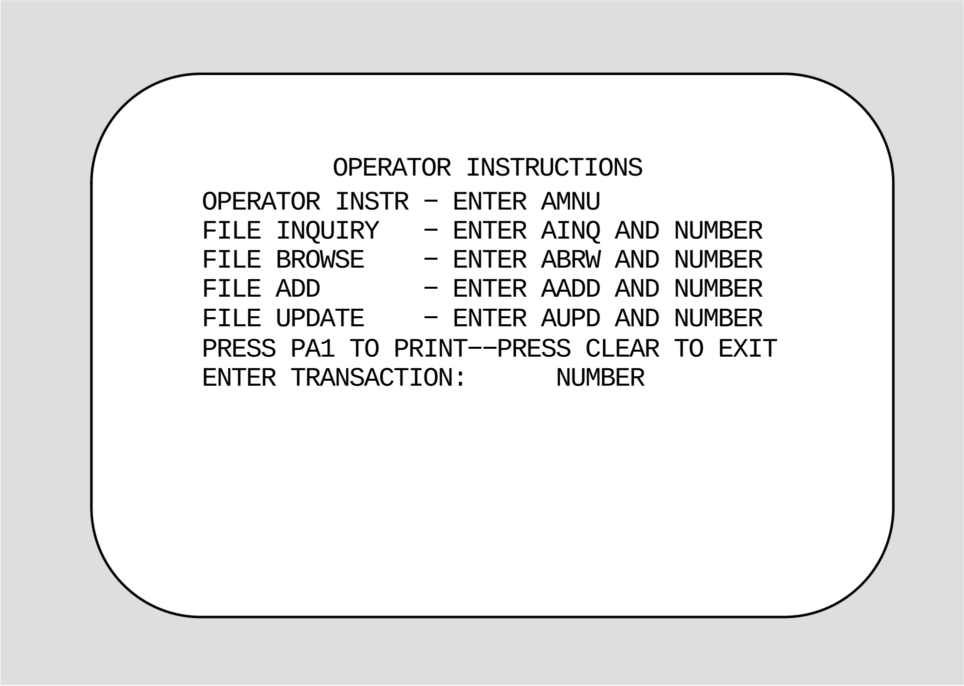
AMNU

LKG−8082−93R

```
RETURN_CODE = SNA3270$WRITE_FIELD(SESSION_ID,
1 %DESCR(STATUS_VECTOR),DATA)
IF (.NOT. RETURN_CODE) THEN
        ISTAT = SYS$PUTMSG(STATUS_VECTOR)
        STOP 'WRITE_FIELD failed'
ENDIF
                .
                .
                .
```

The following code transmits the updated AMNU screen back to the IBM system. IBM responds with a screen that displays operator instructions (see Figure F2H9).

Figure 2–9 AMNU Operator Instructions Screen



```
          OPERATOR INSTRUCTIONS
OPERATOR INSTR − ENTER AMNU
FILE INQUIRY   − ENTER AINQ AND NUMBER
FILE BROWSE    − ENTER ABRW AND NUMBER
FILE ADD       − ENTER AADD AND NUMBER
FILE UPDATE    − ENTER AUPD AND NUMBER
PRESS PA1 TO PRINT−−PRESS CLEAR TO EXIT
ENTER TRANSACTION:      NUMBER
```

LKG−8083−93R

```
RETURN_CODE = SNA3270$RECEIVE_SCREEN_W (SESSION_ID,
1 %DESCR(STATUS_VECTOR), %REF(LU2_EFN))
IF (.NOT. RETURN_CODE) THEN
        ISTAT = SYS$PUTMSG(STATUS_VECTOR)
        STOP 'RECEIVE_SCREEN failed'
ENDIF
                .
                .
                .
```

The following program code transmits the transaction name and number to the IBM application. The field to be located is described as an unprotected field on the operator instructions screen.

```
FDB.SNA3270$W_FDB_ATT_MASK = SNA3270$M_ATTR_PRO
FDB.SNA3270$W_FDB_ATT_VALUE = 0
```

The search begins starting from character position 0.

```
        FDB.SNA3270$W_FDB_ATT_BUFOFF = 0
        FDB.SNA3270$W_FDB_SELECT = SNA3270$K_SEL_SEARCH
```

The first unprotected field on the screen is located.

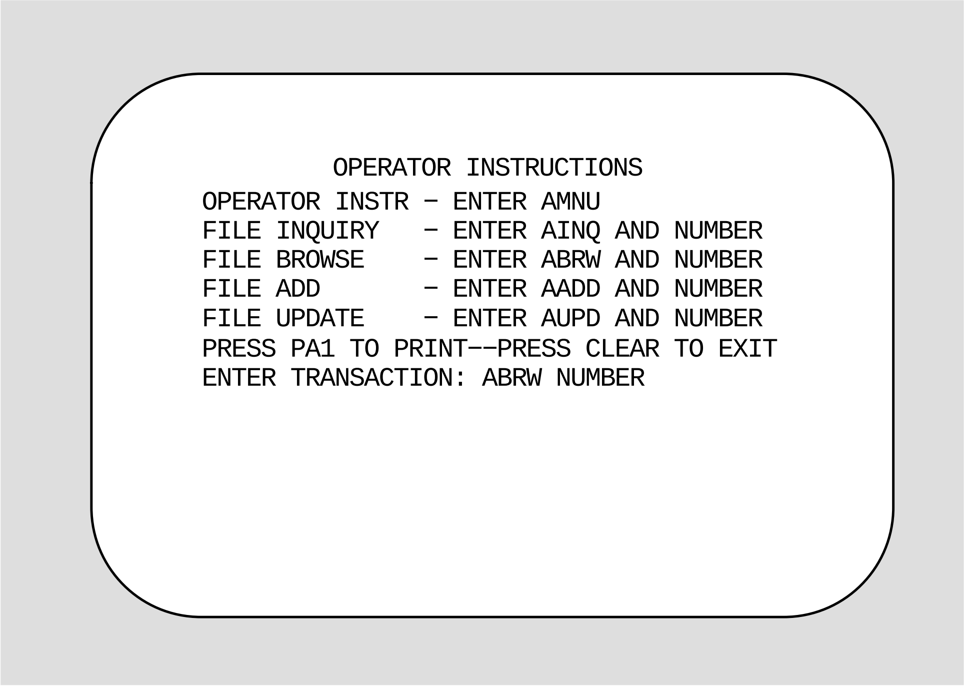
```
        RETURN_CODE = SNA3270$READ_FIELD(SESSION_ID,
        1 %DESCR(STATUS_VECTOR))
        IF (.NOT. RETURN_CODE) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP 'READ_FIELD failed'
        ENDIF
```

ABRW is set as the transaction name, and the string is converted to EBCDIC.

```
        FIELD_DATA = 'ABRW'
        ISTAT = LIB$TRA_ASC_EBC (DATA, DATA)
        IF (.NOT. ISTAT) CALL LIB$STOP(%VAL(ISTAT))
```

The user enters the browse code ABRW into the field just located (see Figure 2–10).

Figure 2–10 ABRW Field Filled on Option Screen



```
         OPERATOR INSTRUCTIONS
OPERATOR INSTR - ENTER AMNU
FILE INQUIRY   - ENTER AINQ AND NUMBER
FILE BROWSE    - ENTER ABRW AND NUMBER
FILE ADD       - ENTER AADD AND NUMBER
FILE UPDATE    - ENTER AUPD AND NUMBER
PRESS PA1 TO PRINT--PRESS CLEAR TO EXIT
ENTER TRANSACTION: ABRW NUMBER
```

LKG−8084−93R

```
RETURN_CODE = SNA3270$WRITE_FIELD(SESSION_ID,
1 %DESCR(STATUS_VECTOR), DATA)

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

In the next code, the number field is described as an unprotected field. Note that the field attribute mask and attribute value have not changed.

```
FDB.SNA3270$W_FDB_ATT_MASK = SNA3270$M_ATTR_PRO
FDB.SNA3270$W_FDB_ATT_VALUE = 0
```

A search for the next unprotected field starts after the last field read.

```
FDB.SNA3270$W_FDB_ATT_SELECT  = SNA3270$K_SEL_SEARCH_NEXT
```

The next unprotected field on the screen is located.

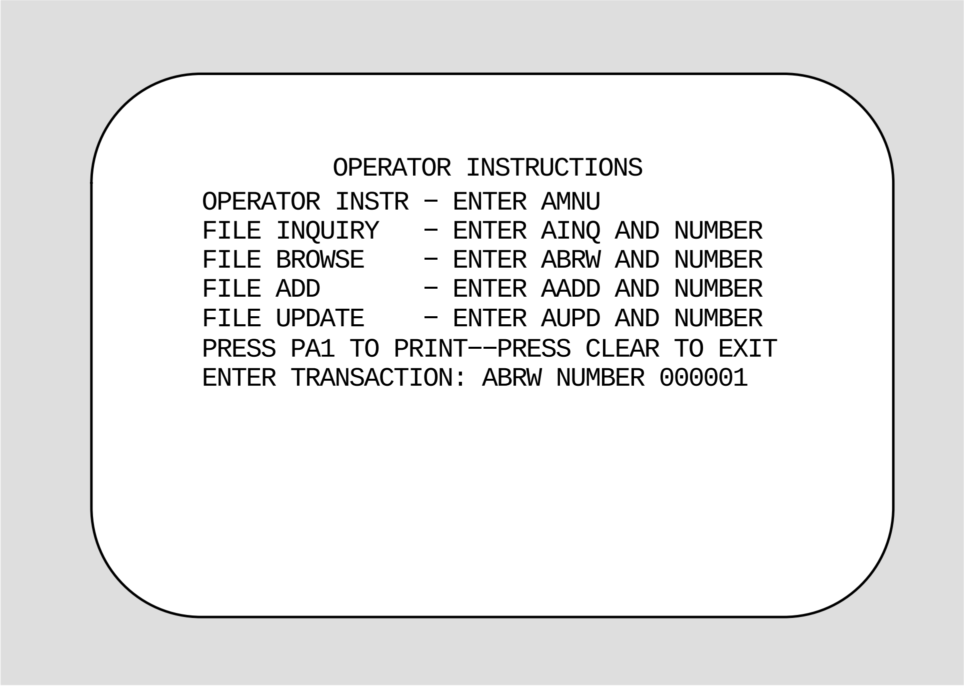
```
RETURN_CODE = SNA3270$READ_FIELD(SESSION_ID,
1 %DESCR(STATUS_VECTOR))

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

The program sets the the transaction number to 000001 and converts the string to EBCDIC.

```
FIELD_DATA = '000001'
ISTAT = LIB$TRA_ASC_EBC (DATA, DATA)
IF (.NOT. ISTAT) CALL LIB$STOP(%VAL(ISTAT))
```

The transaction number 000001 is written into the field just located (see Figure 2–11).

Figure 2–11 Transaction Number Field Filled on Option Screen



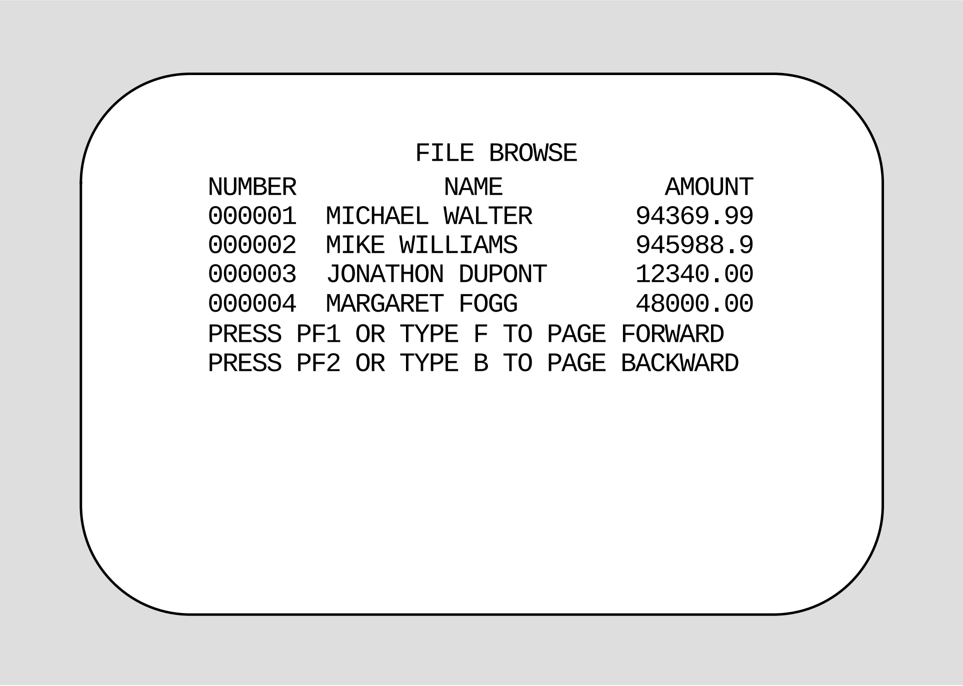
```
          OPERATOR INSTRUCTIONS
OPERATOR INSTR - ENTER AMNU
FILE INQUIRY   - ENTER AINQ AND NUMBER
FILE BROWSE    - ENTER ABRW AND NUMBER
FILE ADD       - ENTER AADD AND NUMBER
FILE UPDATE    - ENTER AUPD AND NUMBER
PRESS PA1 TO PRINT--PRESS CLEAR TO EXIT
ENTER TRANSACTION: ABRW NUMBER 000001
```

LKG−8085−93R

```
RETURN_CODE = SNA3270$WRITE_FIELD(SESSION_ID,
1  %DESCR(STATUS_VECTOR), DATA)
IF (.NOT. RETURN_CODE) THEN ISTAT = SYS$PUTMSG(STATUS_VECTOR)
   STOP 'WRITE_FIELD failed'
ENDIF

                .
                .
                .
```

The completed options screen is sent to IBM, and IBM responds with the screen shown in Figure 2–12.

Figure 2–12 AMNU File Browse Screen



```
              FILE BROWSE
NUMBER          NAME           AMOUNT
000001  MICHAEL WALTER       94369.99
000002  MIKE WILLIAMS        945988.9
000003  JONATHON DUPONT      12340.00
000004  MARGARET FOGG        48000.00
PRESS PF1 OR TYPE F TO PAGE FORWARD
PRESS PF2 OR TYPE B TO PAGE BACKWARD
```

LKG−8086−93R

2.4.7 Writing a Field Into a 3270 Screen Image

To write a field into the display vectors, the OpenVMS application calls the SNA3270$WRITE_FIELD procedure, specifies the buffer containing the field, and uses the field descriptor block to specify the address of the field in the actual displacement field (SNA3270$_FDB_FLD_BUFOFF). Use the SNA3270$READ_FIELD procedure to set up the FDB for the SNA3270$WRITE_FIELD. In particular, SNA3270$READ_FIELD sets the actual attributes and the actual displacement which are inputs to the SNA3270$WRITE_FIELD. The Interface places the contents of the buffer in the specified field, unless that field is protected or of a different data type. If the specified data is larger than the field in the screen image, the procedure returns the SNA3270$_OK_TRUNC status value.

2.4.8 Transmitting a 3270 Screen Image

To transmit a 3270 screen image to the IBM host, the OpenVMS application calls the SNA3270$TRANSMIT_SCREEN procedure and specifies a session. The Interface builds a 3270 data stream from the contents of the display vectors assigned to the session and transmits the data stream to the IBM host.

Note

In an LU2 session, the OpenVMS application normally alternates between receiving screens and sending screens. If the application issues two SNA3270$TRANSMIT_SCREEN requests in a row, the Interface returns the status message SNA3270$_NYTXMIT, indicating that it is "Not your turn." The OpenVMS application can use the SNA3270$TRANSMIT_SIGNAL procedure described in Section 6.10 to request the IBM application to allow the OpenVMS application to transmit again. As described in Section 2.3.1, the OpenVMS application must wait for the IBM application to return the flag indicating who can send before issuing another SNA3270$TRANSMIT_SCREEN.

2.5 Terminating a Session

To terminate an LU-LU session, the OpenVMS application calls the SNA3270$REQUEST_DISCONNECT procedure and specifies the session.

As a result of this call, the Interface requests termination and deallocates all resources allocated to the session. The SNA3270$REQUEST_DISCONNECT procedure causes the Interface to send an UNBIND to the IBM system and results in a log message on the IBM system. SNA3270$REQUEST_DISCONNECT does not perform an orderly shutdown of the session and/or log off the IBM subsystem. If the session is already inactive when the application calls the SNA3270$REQUEST_DISCONNECT procedure, the Interface just deallocates resources.

3 Interface Features

The Digital SNA 3270 Data Stream Programming Interface provides features to assist you in writing and executing your application. These features include mechanisms for:

- Returning status information
- Asynchronous event notification
- Synchronous and asynchronous operation
- Supplying access information to the IBM host

3.1 Returning Status Information

The Interface can return status codes to the OpenVMS application with either of the following:

- Function value returns
- An I/O status vector

See Appendix D for a description of all the status codes returned by the Interface.

3.1.1 Function Value Returns

When an Interface procedure finishes its attempt to perform an operation, it returns a function value to indicate whether the operation succeeded or failed. It places this value in register R0. After each call to an Interface procedure, you must check this status value. The value in the low-order word indicates 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 Interface procedures return status messages to the OpenVMS application via a status vector. The status vector is a data structure that supplies the application with complete information about error conditions, including:

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

The status vector can contain one or more error messages, depending upon the kind of error that occurred. The format of the status vector is identical to the format of the message vector supplied to the OpenVMS system service procedure $PUTMSG.

Note

If a procedure returns a status message, a violation may have occurred that requires action. For example, if you issue the SNA3270$RECEIVE_STREAM procedure and receive the SNA3270$_OK_NYT success message, you cannot transmit. Rather, you must post another receive.

3.1.3 Using Status Vectors

If an error occurs, each component of the network involved can pass a message to the Interface. The Interface uses this information to fill a status vector supplied by the OpenVMS application. See Figure 3–1 for an illustration of the status vector.

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.

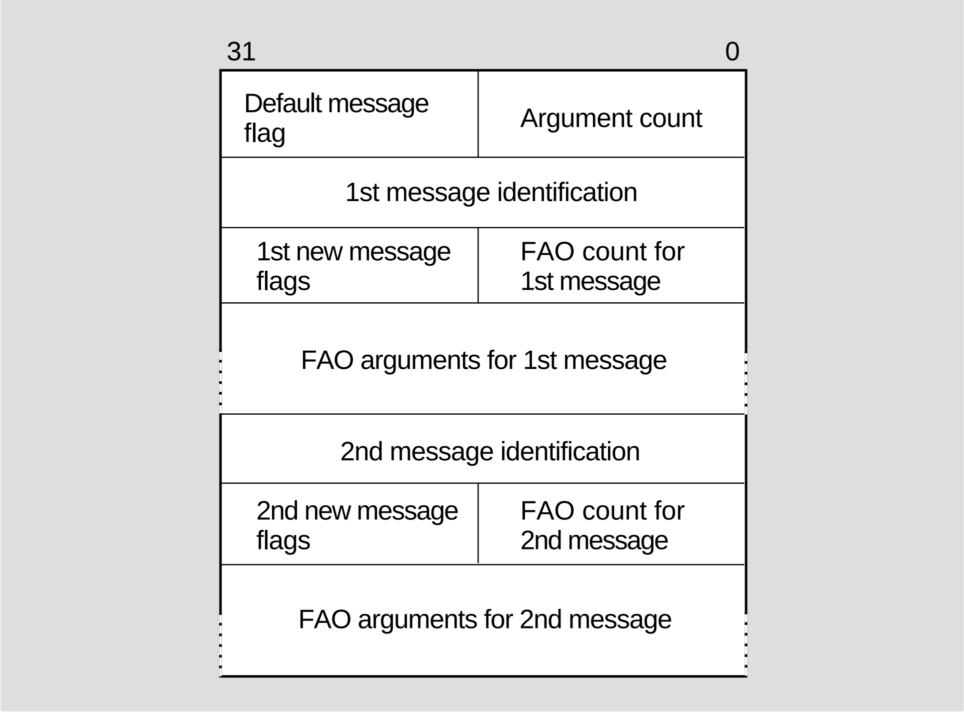
For further information about dealing with errors, see under "Condition Handling" and "$PUTMSG" in the *OpenVMS System Services Reference Manual*. See under "LIB$CALLG", "LIB$SIGNAL", and "LIB$STOP" in the *RTL Library (LIB$) Manual*.

The application does not have to signal an error through $PUTMSG. Instead, the programmer may choose to take corrective action. For example, the programmer may wish to reestablish a session if the session was inadvertently terminated.

Note

The user must define a vector of minimum size, using the SNA3270$K_MIN_STATUS_VECTOR literal, and provide a descriptor pointing to it in each procedure call. The Interface can then fill in the vector at the completion of the operation.

Figure 3–1 Status Vector



31	0
Default message flag	Argument count
1st message identification	
1st new message flags	FAO count for 1st message
FAO arguments for 1st message	
2nd message identification	
2nd new message flags	FAO count for 2nd message
FAO arguments for 2nd message	

LKG−8087−93R

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

- **Argument count**

  Specifies the total number of longwords in the status vector.

- **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 $GETMSG as the FLAGS argument. For further information, see under "$GETMSG" 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 specified new message flags field 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 ($FAO) arguments for this message, if any, that follow in the status vector. For further information see under "$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 Asynchronous Event Notification

The 3270 Interface 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 or the SNA Gateway has deliberately terminated the link.
- The IBM host has violated the SNA protocol.
- The IBM host has sent an UNBIND type 2.
- The session has been reconnected–resume normal operations.
- The session has received a CLEAR.

- A protocol processing error has been detected.
- The PLU has taken the SLU’s turn to send.
- Data has arrived and there is no receive pending from the application. This notification takes place only when the OpenVMS and IBM applications are in contention state (that is, either the OpenVMS or IBM application can transmit).

The OpenVMS application can include a user-written notification procedure that the Interface calls each time one of these asynchronous events occurs during a session. When you call SNA3270$REQUEST_CONNECT, use the *notify-rtn* parameter to indicate the user-written procedure. The notify routine can examine the event code it receives and take action, such as return 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 (*event-code.rz.r,notify-parm.rlu.r*)

where

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

event-code — is a symbol for the asynchronous event indicating the nature of the event.

- SNA3270$K_EVT_TERMINATE–A deliberate termination of the link by the IBM host or the SNA Gateway has occurred. This is a fatal error; you must reestablish the session.
- SNA3270$K_EVT_UNBINDT2–An UNBIND type 2 has been sent by IBM. The session should be automatically reestablished by the Interface. See the RECONNECTED event below.
- SNA3270$K_EVT_RECONNECTED–The session is ready to resume normal operations after an UNBIND type 2.
- SNA3270$K_EVT_CLEAR–The session has received a CLEAR sent by primary session control to reset the data traffic Finite State Machines (FSMs) in the primary and secondary half-sessions (and boundary function, if any). Reset the session and resume normal data.
- SNA3270$K_EVT_PROPROERR–A protocol processing error has been detected. This is a fatal error; you must reestablish the session.
- SNA3270$K_EVT_DATA–Data has arrived and there is no corresponding receive pending from the application.

- SNA3270$K_EVT_TURNGONE–The SLU's turn to send was taken by the PLU. The application should issue a SNA3270$RECEIVE_STREAM or SNA3270$RECEIVE_SCREEN, depending on the connection mode.

notify-parm is an optional user-specified parameter to be passed to the notification procedure. For example, 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 in the preceding list occurs, the following steps take place:

1. The Interface fills out the notify vector supplied by the OpenVMS application in the SNA3270$REQUEST_CONNECT procedure.
2. The Interface notifies the OpenVMS application of the asynchronous event by calling the user-written notify procedure.
3. The OpenVMS application reads the event code supplied 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 is the same as that of the status vector. See Section 3.1.2 for further information.

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

The user must define a vector of minimum size, using the SNA3270$K_MIN_STATUS_VECTOR literal, and provide a descriptor pointing to it in the SNA3270$REQUEST_CONNECT procedure call. The Interface can then fill in the vector.

3.3 Synchronous and Asynchronous Operation

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

3.3.1 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 Interface returns status information immediately as a function value and with further information in the status vector.
3. The Interface sends the request to the Gateway and suspends the OpenVMS application.
4. The Gateway performs the operation and sends the result to the Interface.
5. The Interface 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:

SNA3270$*procedure*_W (*parameters*)

where

SNA3270$procedure_W	is the name of the procedure.
parameters	is a list of information needed to perform the requested operation.

3.3.2 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 specify an event flag or provide a completion procedure that the Interface can call to indicate that the Gateway has completed its attempt to perform the operation.

The 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

The Interface 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 Interface 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 Interface procedure returns a function value indicating success of the call, and the application resumes execution. At the completion of the operation, the Interface 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 to inform the application that the Interface has finished its attempt to perform the requested operation
4. The Interface 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 Interface does not attempt to perform the operation. The application resumes execution.

Note

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

An asynchronous call has the following general format:

SNA3270$*procedure* (*parameters*)

where

SNA3270$procedure — is the name of the procedure.

parameters — is a list of information needed to perform the requested operation. The user-written procedure that an Interface procedure calls to indicate that the Interface 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 Interface 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 Interface 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 Interface procedure. If the application omits the event flag number, the Interface assumes event flag 0.

3.4 Supplying Access Information to the IBM Host

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

- **Physical Unit (PU) identification.** A value identifying the DECnet Gateway PU or OpenVMS SNA (for example, SNA-0) used to establish the session. This information can only be supplied to the DECnet SNA Gateways and OpenVMS SNA. It is replaced by the Logical Unit Identifaction for the Digital SNA Domain Gateway and the Digital SNA Peer Server.
- **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 that you want to use to establish a session with the IBM host. This information can only be supplied to the DECnet SNA Gateways and OpenVMS SNA. It is ignored for the Domain Gateway and the Peer Server.
- **Logical Unit (LU) identification.** A value identifying the Gateway LU used to establish the session. This information can only be supplied to the Domain Gateway and Peer Server.
- **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.
- **IBM password.** A string associated with the IBM user ID. (Some IBM applications require a password, others do not.)
- **Optional user data.** Data passed to the IBM application. (The meaning of the data depends on the IBM application.)

The application supplies this information as parameters each time it issues a call to the SNA3270$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 SNA3270$REQUEST_CONNECT, all IBM access information defaults to the values in the associated list. To override a value associated with an access name, specify a new value in the SNA3270$REQUEST_CONNECT call. For further information about IBM access information and access names, see the *Digital SNA Domain Gateway Management*, *DECnet SNA Gateway Management for OpenVMS*, *OpenVMS SNA Management*, or the *DECnet SNA Gateway-CT Management (OpenVMS)*.

4

Linking an Application With the 3270 Data Stream Interface

After you have written your application and compiled the modules, you are ready to link them with the Interface procedures. You must use the shareable image of Interface procedures to link your program.

The following example links your executable image and the shareable image SYS$SHARE:SNA3270SH.EXE. You must specify a linker options file.

```
$ LINK/EXE/MAP  SYS$INPUT:/OPTION
USERPROG2
SYS$SHARE:SNA3270SH/SHARE
CTLR/Z
$
```

The following example links your executable image with the debugger and the shareable image SYS$SHARE:SNA3270SH.EXE. You must specify a linker options file.

```
$ LINK/EXE/MAP/DEBUG  SYS$INPUT:/OPTION
USERPROG2
SYS$SHARE:SNA3270SH/SHARE
CTLR/Z
$
```

For a detailed description of the LINK command and additional options, see the *OpenVMS Linker Utility Manual*.

5 Programming Examples

This chapter contains programming examples designed to show you how to make calls to the Digital SNA 3270 Data Stream Programming Interface in your OpenVMS applications. Two complete programming examples and fragments of several programs illustrate how to use the Interface in data stream mode and field mode. The programming fragments are provided for information only; they are not complete programs and will not run if you enter them into your system. 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 (a complete field mode example)
- PL/I (a complete data stream mode example)
- C
- COBOL
- BLISS
- MACRO
- Pascal

Explanatory text accompanies each example. The explanations are keyed to the examples by means of a special numeric symbol.

5.1 FORTRAN Programming Example–Field Mode

The FORTRAN program that follows uses field mode in the 3270 Data Stream Interface to connect to the AMNU transaction, a sample program, running under CICS. The program prompts the user for the Gateway DECnet node name or TCP/IP host name and the CICS access name and then connects the OpenVMS application with CICS and requests the AMNU transaction. After the application receives the AMNU operator screen, it requests the browse function of the AMNU program by writing "ABRW" and "000001" into the two unprotected fields of the operator screen.

The OpenVMS application receives the first screen of the browse function and makes successive calls to SNA3270$READ_FIELD to read each field on the screen. Each field is displayed giving its attributes, text, and length. You can use this technique to analyze any screen image that an IBM application might send.

The following list provides a step-by-step description of what this program does:

1. Prompts the user for a Gateway DECnet node name or TCP/IP host name and access name
2. Requests a connection with IBM
3. Receives the CICS logo
4. Sends a clear screen request
5. Receives a clear screen command
6. Requests the AMNU transaction
7. Receives the AMNU operator screen
8. Requests the browse operation
9. Finds the first unprotected field
10. Inserts "ABRW" in the first unprotected field
11. Finds the next unprotected field
12. Inserts "000001" in the next unprotected field
13. Sends the browse request with the AID key equal to enter
14. Receives the browse screen
15. Finds the next field with the unspecified attributes
16. Displays the field number, attributes, size, and contents

17. Successively reads fields until it has displayed them all

18. Issues a disconnect request

```
         PROGRAM LU2_EXAMPLE
 C
    [1]  INCLUDE ’SYS$LIBRARY:SNA3270DF/NOLIST’  ! Include 3270
                                                 ! definitions
 C
 C       define misc. session variables
 C
         INTEGER*4 STATUS_VECTOR (SNA3270$K_MIN_STATUS_VECTOR)
         INTEGER*4 NOTIFY_VECTOR (SNA3270$K_MIN_NOTIFY_VECTOR)
         INTEGER*4 SESSION_ID
         INTEGER*4 RETURN_CODE, CONN_TYPE
         PARAMETER LU2_EFN = 10
         EXTERNAL  NOTIFY_RTN            ! Define notify routine
 C
 C       Define screen image
 C
         INTEGER*4 SCREEN_SIZE           ! Define screen size
    [2]  PARAMETER (SCREEN_SIZE = 3169)
         CHARACTER*(SCREEN_SIZE) SCREEN_IMAGE
 C
 C       Define field mode structures
 C
 C       STRUCTURE /DSC/
 C               INTEGER*2 DSC$W_LENGTH
 C  [3]          BYTE DSC$B_DTYPE, DSC$B_CLASS
 C               INTEGER*4 DSC$A_POINTER
 C       END STRUCTURE
         RECORD /SNA3270_SDB/ SDB
         RECORD /SNA3270_FDB/ FDB
 C       RECORD /DSC/ SDB_DSC
 C       RECORD /DSC/ FDB_DSC
         INTEGER*4 SDB_DSC(2), FDB_DSC(2)
         CHARACTER*256 FIELD_DATA
         INTEGER*2 FIELD_ATTR, FIELD_OFFSET, LENGTH
         LOGICAL*1 FIELD_VECTOR(SCREEN_SIZE/8+1)
         INTEGER*2 ATTR_VECTOR(SCREEN_SIZE)
         CHARACTER*8 NODE_NAME,ACC_NAME
 C
 C       Global data
 C
         COMMON /SESSION_DATA/ SESSION_ID, STATUS_VECTOR
         COMMON /NOTIFY/NOTIFY_VECTOR            ! So notify routine
                                                 ! can access
 C
 C       Initialize FDB and SDB descriptors
 C
         SDB_DSC(1) = SNA3270$K_SDB_LENGTH
    [4]  SDB_DSC(2) = %LOC (SDB)
```

```
        FDB_DSC(1) = SNA3270$K_FDB_SIZE
        FDB_DSC(2) = %LOC (FDB)
C
C       Get gateway or TCP/IP host name and access name
C
        TYPE 9001                       ! Prompt for gateway or TCP/IP host name
   [5]  ACCEPT 9002, NODE_NAME          ! Input gateway or TCP/IP host name
        TYPE 9003                       ! Prompt for access name
   [5]  ACCEPT 9002, ACC_NAME           ! Input access name
C
C       Request Field Mode connection (null parameters are explicit
C       access parameters, access name is used instead)
C
        RETURN_CODE = SNA3270$REQUEST_CONNECT_W (SESSION_ID,
        1 %DESCR(STATUS_VECTOR), %REF(SNA3270$K_ACTIVE),
        2 %REF(SNA3270$K_FIELD_MODE), %DESCR(NODE_NAME),
        3 %DESCR(ACC_NAME),,,,,, [6] SCREEN_IMAGE, [7]
        4 %DESCR(ATTR_VECTOR),
   [8]  5 %DESCR(FIELD_VECTOR), SDB_DSC, FDB_DSC, NOTIFY_RTN,
        6 SESSION_ID, %DESCR(NOTIFY_VECTOR), %REF(LU2_EFN))

        IF (.NOT. RETURN_CODE) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP ’CONNECT failed’
        ENDIF
C
C       Receive CICS logo
C
        RETURN_CODE = SNA3270$RECEIVE_SCREEN_W (SESSION_ID,
        1 %DESCR(STATUS_VECTOR), %REF(LU2_EFN))
        IF (.NOT. RETURN_CODE) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP ’RECEIVE_SCREEN failed’
        ENDIF
C
C       Send clear screen request
C
        RETURN_CODE = SNA3270$TRANSMIT_SCREEN_W (SESSION_ID,
        1 %DESCR(STATUS_VECTOR), %REF(SNA3270$K_AID_CLEAR),
        2 %REF(LU2_EFN))
        IF (.NOT. RETURN_CODE) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP ’TRANSMIT_SCREEN failed’
        ENDIF
C
C       Receive clear screen command
C
        RETURN_CODE = SNA3270$RECEIVE_SCREEN_W (SESSION_ID,
        1 %DESCR(STATUS_VECTOR), %REF(LU2_EFN))
        IF (.NOT. RETURN_CODE) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP ’RECEIVE_SCREEN failed’
        ENDIF

C
C       Transmit transaction name (AMNU)
C

        FIELD_DATA = 'AMNU'
        CALL WRITE_NEXT_FIELD (FIELD_DATA(1:4), FDB)

        RETURN_CODE = SNA3270$TRANSMIT_SCREEN_W (SESSION_ID,
        1 %DESCR(STATUS_VECTOR), %REF(SNA3270$K_AID_ENTER),
        2 %REF(LU2_EFN))
        IF (.NOT. RETURN_CODE) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP 'TRANSMIT_SCREEN failed'
        ENDIF
C
C       Receive AMNU operator screen
C
        RETURN_CODE = SNA3270$RECEIVE_SCREEN_W (SESSION_ID,
        1 %DESCR(STATUS_VECTOR), %REF(LU2_EFN))
        IF (.NOT. RETURN_CODE) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP 'RECEIVE_SCREEN failed'
        ENDIF
C
C       Request browse operation
C
        FIELD_DATA = 'ABRW'
        CALL WRITE_NEXT_FIELD (FIELD_DATA(1:4), FDB)
C
C
C
        FIELD_DATA = '000001'
        CALL WRITE_NEXT_FIELD (FIELD_DATA(1:6), FDB)

        RETURN_CODE = SNA3270$TRANSMIT_SCREEN_W (SESSION_ID,
        1 %DESCR(STATUS_VECTOR), %REF(SNA3270$K_AID_ENTER),
        2 %REF(LU2_EFN))
        IF (.NOT. RETURN_CODE) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP 'TRANSMIT_SCREEN failed'
        ENDIF
C
C       Receive browse screen
C
        RETURN_CODE = SNA3270$RECEIVE_SCREEN_W (SESSION_ID,
        1 %DESCR(STATUS_VECTOR), %REF(LU2_EFN))
        IF (.NOT. RETURN_CODE) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP 'RECEIVE_SCREEN failed'
        ENDIF
```

```
C
C       Read a field
C
   [9]  FDB.SNA3270$W_FDB_ATT_VALUE = SNA3270$M_ATTR_PRO
   [9]  FDB.SNA3270$W_FDB_ATT_MASK = SNA3270$M_ATTR_PRO
        FDB.SNA3270$W_FDB_SELECT = SNA3270$K_SEL_SEARCH_NEXT

  [10] DO 100 I=1,20
  [11] RETURN_CODE = SNA3270$READ_FIELD (SESSION_ID,
        1 %DESCR(STATUS_VECTOR), FIELD_DATA)

        IF (.NOT. RETURN_CODE) THEN
C               IF (STATUS_VECTOR(4) .NE. SNA3270$_RTRUNC) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP 'READ_FIELD failed'
C               ENDIF
        ENDIF
C
C       Get field length, but limit length to maximum output
C       length.
C
        LENGTH = FDB.SNA3270$W_FDB_FLD_COUNT
        IF (LENGTH .GT. 62) LENGTH = 62
C
C       Translate to EBCDIC and display
C
        ISTAT = LIB$TRA_EBC_ASC(FIELD_DATA(:LENGTH),
        1 FIELD_DATA(:LENGTH))
        TYPE 9000, I, FDB.SNA3270$W_FDB_FLD_ATTR,
        1 FDB.SNA3270$W_FDB_FLD_COUNT, FIELD_DATA(:LENGTH)
100     CONTINUE
C
C       Disconnect session
C
        RETURN_CODE = SNA3270$REQUEST_DISCONNECT (SESSION_ID,
        1 %DESCR(STATUS_VECTOR))
        IF (.NOT. RETURN_CODE) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP 'DISCONNECT failed'
        ENDIF

        STOP 'End of LU2 example'
C
C       Format statements
C

9000    FORMAT (1X,I2,2X,Z4,2X,I5,2X,62A)
9001    FORMAT (1X,'Enter gateway DECnet node name or TCP/IP host name: ',2X,$)
9002    FORMAT (A8)
9003    FORMAT (1X,'Enter access name: ',2X,$)

        END

        SUBROUTINE NOTIFY_RTN (EVENT_CODE, NOTIFY_PARM)
```

```
C**********************************************************************
C                                                                     *
C  This is the asynchronous event notification routine. All this      *
C  implementation does is report that an event has occurred.  A more  *
C  complete implementation might take specific action based on the    *
C  event, such as trying to reestablish a session that was terminated *
C  or issuing a receive if the data event was received.               *
C                                                                     *
C**********************************************************************
        INCLUDE ’SYS$LIBRARY:SNA3270DF/NOLIST’  ! Include 3270
                                                ! definitions

        INTEGER*4 NOTIFY_VECTOR (SNA3270$K_MIN_NOTIFY_VECTOR)
        INTEGER*4 EVENT_CODE
        CHARACTER CLEAR_EVENT*14/’Clear received’/
        CHARACTER DATA_EVENT*13/’Data received’/
        CHARACTER PROERR_EVENT*27/’SNA protocol error detected’/
        CHARACTER RECON_EVENT*19/’Session reconnected’/
        CHARACTER TERM_EVENT*18/’Session terminated’/
        CHARACTER TURN_EVENT*15/’PLU RTS honored’/
        CHARACTER UNBIND_EVENT*22/’Unbind type 2 received’/
C
C       Global data
C
        COMMON /SESSION_DATA/ SESSION_ID, STATUS_VECTOR
        COMMON /NOTIFY/NOTIFY_VECTOR            ! So notify routine
                                                ! can access
        IF (EVENT_CODE .EQ. SNA3270$K_EVT_CLEAR) THEN
           TYPE 9100, CLEAR_EVENT
        ENDIF
        IF (EVENT_CODE .EQ. SNA3270$K_EVT_DATA) THEN
           TYPE 9100, DATA_EVENT
        ENDIF
        IF (EVENT_CODE .EQ. SNA3270$K_EVT_PROPROERR) THEN
           TYPE 9100, PROERR_EVENT
        ENDIF
        IF (EVENT_CODE .EQ. SNA3270$K_EVT_RECONNECTED) THEN
           TYPE 9100, RECON_EVENT
        ENDIF
        IF (EVENT_CODE .EQ. SNA3270$K_EVT_TERMINATE) THEN
           TYPE 9100, TERM_EVENT
        ENDIF
        IF (EVENT_CODE .EQ. SNA3270$K_EVT_TURNGONE) THEN
           TYPE 9100, TURN_EVENT
        ENDIF
        IF (EVENT_CODE .EQ. SNA3270$K_EVT_UNBINDT2) THEN
           TYPE 9100, UNBIND_EVENT
        ENDIF
        ISTAT = SYS$PUTMSG(NOTIFY_VECTOR)
        RETURN
9100    FORMAT (3X,'Asynchronous notification: ',A30//)
        END

        SUBROUTINE WRITE_NEXT_FIELD (DATA, FDB)
C**********************************************************************
C                                                                     *
C  This routine first calls SNA3270$READ_FIELD to position the FDB    *
C  pointers to the next unprotected field.  It then translates the    *
C  data into EBCDIC and calls SNA3270$WRITE_FIELD to write the data   *
C  into the screen image.                                             *
C                                                                     *
C**********************************************************************

        INCLUDE 'SYS$LIBRARY:SNA3270DF/NOLIST'

        CHARACTER*(*) DATA
        INTEGER*4 STATUS_VECTOR (SNA3270$K_MIN_STATUS_VECTOR)
        INTEGER*4 SESSION_ID
        INTEGER*4 RETURN_CODE
        RECORD /SNA3270_FDB/ FDB

        COMMON /SESSION_DATA/ SESSION_ID, STATUS_VECTOR

C
C       Find the next unprotected field
C
        FDB.SNA3270$W_FDB_ATT_VALUE = 0
        FDB.SNA3270$W_FDB_ATT_MASK = SNA3270$M_ATTR_PRO
        FDB.SNA3270$W_FDB_BUFOFF = 0
        FDB.SNA3270$W_FDB_SELECT = SNA3270$K_SEL_SEARCH_NEXT
        RETURN_CODE = SNA3270$READ_FIELD (SESSION_ID,
        1 %DESCR(STATUS_VECTOR))

        IF (.NOT. RETURN_CODE) THEN
                ISTAT = SYS$PUTMSG(STATUS_VECTOR)
                STOP 'READ_FIELD failed'
        ENDIF
C
C       Convert string to EBCDIC
C
        ISTAT = LIB$TRA_ASC_EBC (DATA, DATA)
   [12] IF (.NOT. ISTAT) CALL LIB$STOP(%VAL(ISTAT))
C
C       Update the screen image
C
        RETURN_CODE = SNA3270$WRITE_FIELD(SESSION_ID,
        1 %DESCR(STATUS_VECTOR),DATA)

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

        RETURN
        END
```

Comments

1. Include the 3270 Library.
2. The screen image must be one element greater than the larger of the default and alternate screen size (3168 + 1). The first byte is reserved.
3. In FORTRAN, the built-in descriptor mechanism does not allow you to pass a structure by descriptor. To solve this problem, add your own statements to build a structure defining the descriptor, fill in the descriptor and then pass it by reference.
4. The filled-in descriptors.
5. The input for NODE_NAME and ACC_NAME is case sensitive. Use the case appropriate for your needs.
6. Commas indicate that you do not want to specify values for the parameters and will accept the default values provided by the Interface.
7. The SCREEN_IMAGE is defined as a character string and passed by descriptor by default.
8. The SDB_DSC and FDB_DSC are passed by reference, the default for structures.

9. If you specify 0 (no preference) for the attributes, the Interface locates the next field no matter what the attribute.
10. The number of loop counts (for example, 20) you can do is installation dependent.
11. The READ_FIELD procedure always completes synchronously even though no _W is present.
12. You can use OpenVMS Library routines to do parts of your application, such as translating ASCII to EBCDIC or vice versa.

5.2 PL/I Programming Example–Data Stream Mode

The PL/I application shown here uses stream mode in the 3270 Data Stream Interface to connect to IBM. The program invokes the CSFE transaction (a remote loopback program) running under CICS and transmits a string of data to it. IBM then writes the string back to the OpenVMS application. The application then compares the data that it sent with the data returned by IBM.

The following list provides a step-by-step description of what this program does:

1. Requests a connection with IBM
2. Receives the CICS logo
3. Acknowledges the CICS logo
4. Sends a clear screen request
5. Receives a clear screen command
6. Acknowledges a clear screen command
7. Requests the CSFE transaction
8. Receives the CSFE instruction screen
9. Acknowledges the CSFE instruction screen
10. Sends the data string to CSFE
11. Receives data string from CSFE
12. Acknowledges receiving data from CSFE
13. Compares CSFE data with data OpenVMS application originally sent
14. Issues a disconnect request

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

   %INCLUDE $STSDEF;                        /* System status codes  */
   %INCLUDE SYS$PUTMSG;                     /* System Service       */
[1]%INCLUDE ’SYS$LIBRARY:SNA3270DF.PLI’;    /* SNA3270 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));

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

   %REPLACE TEXT_SIZE BY 56;

   DECLARE NODE_NAME CHARACTER (6),
           NODE_NAME_SIZE FIXED BIN (15),
           NODE_PROMPT CHARACTER (18)
                   STATIC INITIAL('Enter node name: '),

           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 (31),
           STATUS_VECTOR CHARACTER (SNA3270$K_MIN_STATUS_VECTOR),

           NOTIFY_VECTOR GLOBALDEF
                   CHARACTER (SNA3270$K_MIN_NOTIFY_VECTOR),

           I FIXED BINARY (8) INITIAL (0),
           LOOP_COUNT FIXED DECIMAL (3) INITIAL (0),
           ERROR_COUNT FIXED BINARY (16) INITIAL (0),

           ASCII_TEXT CHARACTER (TEXT_SIZE) INITIAL
               [2] ('..........
               [3] ABCDEFGHIJKLMNOPQRSTUVWXYZ
                   1234567890.,?/$*'),

           DATA_BUFFER_PTR_1 POINTER,
       [4] DATA_BUFFER_1 CHARACTER (2048),
           DATA_BUFFER_ARRAY_1 (2048)
                   CHARACTER BASED (DATA_BUFFER_PTR_1),

           DATA_BUFFER_PTR_2 POINTER,
       [5] DATA_BUFFER_2 CHARACTER (TEXT_SIZE),
           DATA_BUFFER_ARRAY_2 (TEXT_SIZE)
                   CHARACTER BASED (DATA_BUFFER_PTR_2);
```

```
DATA_BUFFER_PTR_1 = ADDR (DATA_BUFFER_1);
DATA_BUFFER_PTR_2 = ADDR (DATA_BUFFER_2);

/****************************************************************/
/*                                                              */
/*    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$VALUE = SNA3270$REQUEST_CONNECT_W (
                                SESSION_ID,
                                STATUS_VECTOR,
                                SNA3270$K_ACTIVE,
                                SNA3270$K_STREAM_MODE,
                                NODE_NAME,
                                ACCESS_NAME,
                            [6] ,,,,,,,,,,,,
                                ADDR(EXAMPLE$NOTIFY),
                                ,
                                NOTIFY_VECTOR,
                                ,,,);
IF ^STS$SUCCESS THEN GOTO ERROR_FROM_INTERFACE;

/****************************************************************/
/*                                                              */
/*      Read in normal data, this should be the CICS logo       */
/*                                                              */
/****************************************************************/

STS$VALUE = EXAMPLE$READ_STREAM (
                                SESSION_ID,
                                STATUS_VECTOR,
                                DATA_BUFFER_1,
                                DATA_SIZE);

IF ^STS$SUCCESS THEN GOTO ERROR_FROM_INTERFACE;

/****************************************************************/
/*                                                              */
/*       Clear the screen so CICS will restore keyboard         */
/*                                                              */
/****************************************************************/

DATA_BUFFER_ARRAY_2 (8) = BYTE (SNA3270$K_AID_CLEAR);
```

```
[7]DATA_SIZE = SNA3270$K_BUF_HDLEN + 10;

   STS$VALUE  =  SNA3270$TRANSMIT_STREAM_W (
                                   SESSION_ID,
                                   STATUS_VECTOR,
                                   DATA_BUFFER_1,
                                   DATA_SIZE,
                                   SNA3270$K_END_OF_DATA,
                                   ,,     /* EFN,AST,Parm */
                                   );
   IF ^STS$SUCCESS THEN GOTO ERROR_FROM_INTERFACE;

   STS$VALUE = EXAMPLE$READ_STREAM (
                                   SESSION_ID,
                                   STATUS_VECTOR,
                                   DATA_BUFFER_1,
                                   DATA_SIZE);

   IF ^STS$SUCCESS THEN GOTO ERROR_FROM_INTERFACE;

   /****************************************************************/
   /*                                                              */
   /*                 Run the "CSFE" transaction                   */
   /*                                                              */
   /****************************************************************/

   DATA_BUFFER_ARRAY_1 (8) = BYTE(SNA3270$K_AID_ENTER);    /* AID  */
   DATA_BUFFER_ARRAY_1 (9) = BYTE (40);                    /*Cursor*/
   DATA_BUFFER_ARRAY_1 (10)= BYTE (40);                    /* Addr */
   DATA_BUFFER_ARRAY_1 (11) = BYTE (131);                  /*  c   */
   DATA_BUFFER_ARRAY_1 (12) = BYTE (162);                  /*  s   */
[8]DATA_BUFFER_ARRAY_1 (13) = BYTE (134);                  /*  f   */
   DATA_BUFFER_ARRAY_1 (14) = BYTE (133);                  /*  e   */
   DATA_SIZE = SNA3270$K_BUF_HDLEN + 7;

   STS$VALUE  =  SNA3270$TRANSMIT_STREAM_W (
                                   SESSION_ID,
                                   STATUS_VECTOR,
                                   DATA_BUFFER_1,
                                   DATA_SIZE,
                                   SNA3270$K_END_OF_DATA,
                                   ,,              /* EFN,AST,Parm */
                                   );
   IF ^STS$SUCCESS THEN GOTO ERROR_FROM_INTERFACE;

   STS$VALUE = EXAMPLE$READ_STREAM (
                                   SESSION_ID,
                                   STATUS_VECTOR,
                                   DATA_BUFFER_1,
                                   DATA_SIZE);

   IF ^STS$SUCCESS THEN GOTO ERROR_FROM_INTERFACE;
   /****************************************************************/
   /*                                                              */
   /*      Loop a block of data back and forth. Check integrity    */
   /*      of data after each receive. Print statistics and quit.  */
   /*                                                              */
   /****************************************************************/

[9]STS$VALUE = LIB$TRA_ASC_EBC (
                           ASCII_TEXT,
                           DATA_BUFFER_2
                           );

   DATA_BUFFER_ARRAY_1 (8) = BYTE(SNA3270$K_AID_ENTER);    /* AID  */
[10]DATA_BUFFER_ARRAY_1 (9) = BYTE (40);                    /*Cursor*/
[10]DATA_BUFFER_ARRAY_1 (10)= BYTE (40);                    /* Addr */

   DO LOOP_COUNT = 1 TO 10;

       DATA_SIZE = TEXT_SIZE;
       STS$VALUE  =  SNA3270$TRANSMIT_STREAM_W (
                                   SESSION_ID,
                                   STATUS_VECTOR,
                                   DATA_BUFFER_2,
                                   DATA_SIZE,
                                   SNA3270$K_END_OF_DATA,
                                   ,,              /* EFN,AST,Parm */
                                   );
       IF ^STS$SUCCESS THEN GOTO ERROR_FROM_INTERFACE;

       STS$VALUE = EXAMPLE$READ_STREAM (
                               SESSION_ID,
                               STATUS_VECTOR,
                               DATA_BUFFER_1,
                               DATA_SIZE);

       IF ^STS$SUCCESS THEN GOTO ERROR_FROM_INTERFACE;

  [11] DO I = SNA3270$K_BUF_HDLEN + 4 TO TEXT_SIZE;
        IF DATA_BUFFER_ARRAY_1 (I-1) ^= DATA_BUFFER_ARRAY_2 (I) THEN
        DO;
           PUT SKIP LIST (’Error in data byte ’,I,’ on pass ’,LOOP_COUNT);
           PUT SKIP LIST (’   Expected to find ’, DATA_BUFFER_ARRAY_1(I));
           PUT SKIP LIST (’   But found instead ’,DATA_BUFFER_ARRAY_2(I+1));
           ERROR_COUNT = ERROR_COUNT + 1;
        END;
     END;
   END;

   STS$VALUE = SNA3270$REQUEST_DISCONNECT (
                                   SESSION_ID,
                                   STATUS_VECTOR,
                                   ,,
                                   );

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

```
 PUT SKIP (2) LIST (’Exiting after ’,LOOP_COUNT-1,’ passes with ’
                 , ERROR_COUNT, ’ errors’);
 RETURN (1);

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

 /****************************************************************/
 /*                                                              */
 /*              Receive data and acknowledge it                 */
 /*                                                              */
 /****************************************************************/

 EXAMPLE$READ_STREAM:    PROCEDURE (
                             SESSION_ID,
                             STATUS_VECTOR,
                             DATA_BUFFER,
                             DATA_SIZE)
                         RETURNS (FIXED BIN);

 %INCLUDE $STSDEF;                       /* System status codes  */
 %INCLUDE ’SYS$LIBRARY:SNA3270DF.PLI’;   /* SNA3270 symbols and  */
                                         /*  routine definitions */
 DECLARE SESSION_ID FIXED BIN (31),
         DATA_BUFFER CHARACTER (*),
         DATA_SIZE FIXED BIN (31),
         STATUS_VECTOR CHARACTER (*);

 STS$VALUE  =  SNA3270$RECEIVE_STREAM_W (
                                 SESSION_ID,
                                 STATUS_VECTOR,
                                 DATA_BUFFER,
                                 DATA_SIZE,
                                 ,,              /* AST,EFN,Parm */
                                 );

 IF STS$VALUE = 1
 THEN
     DO

[12] STS$VALUE  =  SNA3270$ACKNOWLEDGE (
                                 SESSION_ID,
                                 STATUS_VECTOR,
                                 SNA3270$K_ACK_ACCEPT
                                 );

     END;

 RETURN (STS$VALUE);

 END;
```

```
/****************************************************************/
/*                                                              */
/*              Asynchronous notify routine                     */
/*                                                              */
/****************************************************************/

EXAMPLE$NOTIFY:         PROCEDURE (
                            EVENT_CODE,
                            EVENT_PARAMETER); [13]

%INCLUDE $STSDEF;                      /* System status codes  */
%INCLUDE SYS$PUTMSG;                   /* System Service       */
%INCLUDE 'SYS$LIBRARY:SNA3270DF.PLI';  /* SNA3270 symbols and  */
                                       /*  routine definitions */
DECLARE EVENT_CODE FIXED BINARY (31),
        EVENT_PARAMETER FIXED BINARY (31),
        NOTIFY_VECTOR GLOBALREF CHARACTER;

/****************************************************************/
/*                                                              */
/*      Ignore "Data Arrived" events, display all others       */
/*                                                              */
/****************************************************************/

IF EVENT_CODE ^= SNA3270$K_EVT_DATA
THEN
    STS$VALUE = SYS$PUTMSG (NOTIFY_VECTOR);

END;
```

Comments

1. Include the 3270 Library.
2. The periods are place holders for Interface headers (7 bytes for the Interface header and 3 bytes for the data stream header).
3. Test string is being sent to IBM.
4. Input buffers. DATA_BUFFER_1 receives all data (e.g., the CICS logo).
5. Output buffers. DATA_BUFFER_2 is used to build and transmit the data image.
6. Commas indicate that you do not want to specify values for the parameters and will accept the default values provided by the Interface.
7. The application must leave room in the buffer for header information.
8. CSFE is the remote loopback program running under CICS on IBM.
9. You can use OpenVMS Library routines to do parts of your application, such as translating ASCII to EBCDIC or vice versa.

10. The cursor address is encoded. For information about these codes, see *IBM 3270 Information Display System Data Stream Programmer's Reference*, Order No. GA23-0059.

11. Note that the input header (4 bytes) and output header (3 bytes) have different lengths. Three bytes in both the input and output headers contain 3270 data stream control characters (AID key and cursor address). The fourth byte of the input header contains the write control character. This code compares the data the OpenVMS application sent with the data returned by IBM and diagnoses any errors.

12. The SNA3270$ACKNOWLEDGE procedure would normally be called after examination of the data stream header. The application would reject any unsupported or illegal data stream control characters.

13. The asynchronous notify routine, notify parameter, and notify vector are specified in the REQUEST_CONNECT procedure. If you are using multiple sessions, specify a *session-id* or an internal session data structure in the *event-parameter*, so you can identify a particular session. For more information, see Section 3.2.

5.3 C Programming Example–Data Stream Mode

The C program fragment shown here initiates a session with CICS and then clears the screen. It then terminates the session. You can use this program to verify that you can connect with IBM.

```
   #include "sys$library:descrip.h"    /* Descriptor definitions  */
   #include "sys$library:ssdef.h"      /* System services         */
   #include "sys$library:stsdef.h"
[1]#include "sys$library:sna3270df.h"  /* 3270 library            */

   int

           sense_code,
           status;

   unsigned int
           status_vec[SNA3270$K_MIN_STATUS_VECTOR],
           notify_vec[SNA3270$K_MIN_NOTIFY_VECTOR],
           session_id = 0,
           end_chain = SNA3270$K_END_OF_DATA,
           conn_typ = SNA3270$K_ACTIVE,
           mode_typ = SNA3270$K_STREAM_MODE;

   short unsigned int
           data_length;

   struct  {
           char sna_header [SNA3270$K_BUF_HDLEN]; [2]
           char sna_data [2000];
           } db;

   static  $DESCRIPTOR(node_dsc, "BOOJUM");
   static  $DESCRIPTOR(acc_name_dsc, "XCICS");

   struct  dsc$descriptor
           status_vec_dsc  = {SNA3270$K_MIN_STATUS_VECTOR,0,
                                                   0,status_vec},
           notify_vec_dsc  = {SNA3270$K_MIN_NOTIFY_VECTOR,0,
                                                   0,notify_vec},
                           .
                           .
                           .
   /*                                                            */
   /*        Start by bringing up a session, i.e. connect to CICS */
   /*                                                            */
```

```
status = SNA3270$REQUEST_CONNECT_W(&session_id,
                                &status_vec_dsc,
                                &conn_typ,
                                &mode_typ,
                                &node_dsc,
                                &acc_name_dsc,
                                0,0,0,0,0,0,0,0,0,0,0,0, [3]
                                &notify_rtn,
                                0,
                                &notify_vec_dsc,
                                0,0,0
                                );
        if (!(status & STS$M_SUCCESS)) {
                        .
                        .
                        .
/*                                                             */
/*                      Clear the screen                       */
/*                                                             */
data_length = SNA3270$K_BUF_HDLEN + 1;
db.sna_data[0] = SNA3270$K_AID_CLEAR;
status = SNA3270$TRANSMIT_STREAM_W(&session_id,
                                &status_vec_dsc,
                                &data_bufr_dsc,
                                &data_length,
                                &end_chain,
                                0,0,0
                                );
         if (!(status & STS$M_SUCCESS)) {
                         goto terminate;
           }
status = SNA3270$RECEIVE_STREAM_W(&session_id,
                                &status_vec_dsc,
                                &data_bufr_dsc,
                                &data_length,
                                0,0,0
                                );
        if (!(status & STS$M_SUCCESS)) {
                        .
                        .
                        .
```

```
/*                                                          */
/*              Done, Disconnect the session                */
/*                                                          */

terminate:
status = SNA3270$REQUEST_DISCONNECT_W(&session_id,
                               &status_vec_dsc,
                               0,0,0
                               );
                    .
                    .
                    .
```

Comments

1. Include the 3270 Library.
2. The application must leave room in the buffer for header information.
3. 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 Interface.

5.4 COBOL Programming Example–Data Stream Mode

This COBOL program fragment initiates a session with CICS. It then invokes the CSFE transaction (a remote loopback program) running under CICS and transmits a string of data to it. After receiving the CSFE screen from IBM, the program terminates the session.

```
  IDENTIFICATION DIVISION.
  PROGRAM-ID. TEST2.
  DATA DIVISION.

  WORKING-STORAGE SECTION.

  01      SS-STATUS             PIC S9(09) COMP.
  01      SESS-ID               PIC 9(08)  COMP.
  01      STATUS-VEC            PIC X(64).
  01      SNA3270$L_ACTIVE      PIC 9(08)  COMP VALUE 1.
  01      SNA3270$L_STREAM_MODE PIC 9(08)  COMP VALUE 2.
  01      NODNAM                PIC X(06)  VALUE SPACES.
[1]       ACCNAM                PIC X(08)  VALUE SPACES.
  01      NOTIFY-RTN-NAME       PIC X(06)  VALUE "NOTIFY".
  01      NOTIFY-RTN-ADDR       PIC 9(09)  COMP.
  01      WS-STATUS             PIC X(10).
  01      NOTIFY-VEC            PIC X(64).
  01      WS-NOTIFY-VEC         PIC X(64).
  01      WS-STATUS-VEC         PIC X(64).

  01      TEMP-DATA-BUF.
          02 TEMP-DATA-BUFFER OCCURS 256 TIMES PIC X.

  01      SNA3270$L_ACK_ACCEPT  PIC 9(08)  COMP VALUE 0.
  01      LAST-FLAG             PIC 9(08)  COMP VALUE 1.

  01      DATA-BUFFER.
          02 WS-DATA-BUF OCCURS 1000 TIMES PIC X(02).

  01      DATA-BUF.
          02 DATA-BUF1          PIC 9(7).
          02 DATA-BUF2.
             03 DATA-BUFFER2 OCCURS 1000 TIMES PIC X.
  01      IDX                   PIC 9(02).
  01      SUB                   PIC 9(02).
  01      BUF-LEN               PIC 9(08)  COMP.
  01      TEST-DATA.
          02 TST-DATA OCCURS 52 TIMES PIC X.
```

```
PROCEDURE DIVISION.
MAIN.

        PERFORM GET-NODE-ACC-NAME.
        PERFORM GET-NOTIFY-RTN-ADDR.
        PERFORM REQUEST-CONNECT.
        PERFORM RECEIVE-CICS.
        PERFORM ACKNOWLEDGE-DATA.
        PERFORM TRANSMIT-CLEAR-SCREEN.
        PERFORM RECEIVE-CLEAR-SCREEN.
    [2] PERFORM ACKNOWLEDGE-DATA.
        PERFORM TRANSMIT-CSFE.
        PERFORM RECEIVE-CSFE.
        PERFORM ACKNOWLEDGE-DATA.
        PERFORM TRANSMIT-DATA.
        PERFORM RECEIVE-DATA.
        PERFORM ACKNOWLEDGE-DATA.
        PERFORM REQUEST-DISCONNECT.
        PERFORM EXIT-PROGRAM.

****************************************************************
* Get the address of the notify routine
****************************************************************
GET-NOTIFY-RTN-ADDR.
        CALL "COB$CALL"
                USING BY DESCRIPTOR NOTIFY-RTN-NAME
                GIVING NOTIFY-RTN-ADDR.
                        .
                        .
                        .
****************************************************************
* Try to bring up a session in stream mode
****************************************************************

REQUEST-CONNECT.
        CALL "SNA3270$REQUEST_CONNECT_W" USING
                           BY REFERENCE SESS-ID,
                           BY DESCRIPTOR STATUS-VEC,
                           BY REFERENCE SNA3270$L_ACTIVE,
                                        SNA3270$L_STREAM_MODE,
                           BY DESCRIPTOR NODNAM, ACCNAM,
                           BY VALUE 0,0,0,0,0,0,0,0,0,0,0,0,
                           BY VALUE NOTIFY-RTN-ADDR,
                           BY VALUE 0,
                           BY DESCRIPTOR NOTIFY-VEC,
                           BY VALUE 0,0,0,
                                        GIVING SS-STATUS.

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

```
*****************************************************************
* Receive the CICS logo
*****************************************************************

RECEIVE-CICS.

        MOVE SPACES TO DATA-BUF.
        MOVE ZEROS  TO BUF-LEN.
        CALL "SNA3270$RECEIVE_STREAM_W" USING
                           BY REFERENCE SESS-ID,
                           BY DESCRIPTOR STATUS-VEC,
                           BY DESCRIPTOR DATA-BUF,
                           BY REFERENCE BUF-LEN,
                           BY VALUE 0,0,0,
                                        GIVING SS-STATUS.

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

*****************************************************************
* Acknowledge data received
*****************************************************************

ACKNOWLEDGE-DATA.

        CALL "SNA3270$ACKNOWLEDGE" USING
                               BY REFERENCE SESS-ID,
                               BY DESCRIPTOR STATUS-VEC,
                               BY REFERENCE SNA3270$L_ACK_ACCEPT,
                         GIVING SS-STATUS.

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

                        .
                        .
                        .
*****************************************************************
* Convert CSFE to hex and transmit
*****************************************************************

TRANSMIT-CSFE.
        MOVE ZEROS               TO DATA-BUF1.
        MOVE SPACES              TO DATA-BUF2.
        MOVE 14                  TO BUF-LEN.
        MOVE 1                   TO IDX.
    [3] MOVE "7D40C483A28685"    TO DATA-BUFFER.
        PERFORM CONVERT-TO-HEX 14 TIMES.
        CALL "SNA3270$TRANSMIT_STREAM_W" USING
                                BY REFERENCE SESS-ID,
                                BY DESCRIPTOR STATUS-VEC,
                                BY DESCRIPTOR DATA-BUF,
                                BY REFERENCE BUF-LEN,
                                BY REFERENCE LAST-FLAG,
                                BY VALUE 0,0,0,
                                    GIVING SS-STATUS.

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

****************************************************************
* Receive CSFE screen
****************************************************************

RECEIVE-CSFE.
        MOVE SPACES TO DATA-BUF.
        MOVE ZEROS  TO BUF-LEN.

        CALL "SNA3270$RECEIVE_STREAM_W" USING
                                BY REFERENCE SESS-ID,
                                BY DESCRIPTOR STATUS-VEC,
                                BY DESCRIPTOR DATA-BUF,
                                BY REFERENCE BUF-LEN,
                                BY VALUE 0,0,0,
                                             GIVING SS-STATUS.

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

                        .
                        .
                        .

****************************************************************
* Disconnect link, we are done with the session
****************************************************************

REQUEST-DISCONNECT.
        CALL "SNA3270$REQUEST_DISCONNECT_W" USING
                                BY REFERENCE SESS-ID,
                                BY DESCRIPTOR STATUS-VEC,
                                BY VALUE 0,0,0,
                                             GIVING SS-STATUS.

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

```
  CONVERT-TO-HEX.
          CALL "LIB$CVT_HTB" USING BY VALUE 2,
                                 BY REFERENCE WS-DATA-BUF(IDX),
                                 BY REFERENCE DATA-BUFFER2(IDX),
                                              GIVING SS-STATUS.
          ADD 1 TO IDX.

[4]TRANSLATE-ASC-EBC.
          CALL "LIB$TRA_ASC_EBC" USING BY DESCRIPTOR TST-DATA(SUB),
                                                DATA-BUFFER2(IDX),
                                              GIVING SS-STATUS.
          ADD 1 TO SUB.
          ADD 1 TO IDX.

  TRANSLATE-EBC-ASC.
          CALL "LIB$TRA_EBC_ASC" USING BY DESCRIPTOR DATA-BUFFER2(IDX),
                                                TEMP-DATA-BUFFER(SUB),
                                               GIVING SS-STATUS.

          ADD 1 TO SUB.
          ADD 1 TO IDX.
```

Comments

1. Define 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 used in this example.
3. This data represents three pieces of information:
   - 7D = an AID key (enter)
   - 40C4 = cursor address
   - 83A28685 = CSFE
4. You can use OpenVMS Library routines to do parts of your application, such as translating ASCII to EBCDIC or vice versa.

5.5 BLISS Programming Example–Field Mode

This BLISS program fragment initiates a session with CICS using field mode. It then invokes the CSFE transaction (a remote loopback program) running under CICS and transmits a string of data to it.

```
  MODULE TESTFM (MAIN = FM$MAIN) =

  BEGIN

[1]REQUIRE 'SYS$LIBRARY:SNA3270DF';
  LIBRARY 'SYS$LIBRARY:STARLET';

  EXTERNAL ROUTINE
          LIB$TRA_ASC_EBC         :ADDRESSING_MODE (GENERAL),
          LIB$TRA_EBC_ASC         :ADDRESSING_MODE (GENERAL),
          LIB$PUT_OUTPUT          :ADDRESSING_MODE (GENERAL),
          LIB$GET_INPUT           :ADDRESSING_MODE (GENERAL);

  FORWARD ROUTINE
          FM$MAIN         : NOVALUE,
          NOTIFY$RTN      : NOVALUE;

  LITERAL
          INPUT_BUFFER_SIZE       = 132;

  GLOBAL
          SESS_ID,
          NOTIFY_VECTOR   : VECTOR [SNA3270$K_MIN_NOTIFY_VECTOR, LONG],
          NOTIFY_DSC      : BLOCK [8, BYTE],
          STATUS_VECTOR   : VECTOR [SNA3270$K_MIN_STATUS_VECTOR, LONG],
          STATUS_DSC      : BLOCK [8, BYTE];

  GLOBAL
          BIND
            NOTIFY_VECTOR_SIZE = SNA3270$K_MIN_NOTIFY_VECTOR,
            STATUS_VECTOR_SIZE = SNA3270$K_MIN_STATUS_VECTOR;

  GLOBAL ROUTINE FM$MAIN  : NOVALUE =

  !
  !
  ! This routine tests the field mode connection.
  !
  !

  BEGIN
```

```
LITERAL
        BUFFER_SIZE        = 132,
        DATA_BUFFER_SIZE   = 1000,
        FIELD_VECTOR_SIZE  = 397,
        CHAR_VECTOR_SIZE   = 3169,
        ATTR_VECTOR_SIZE   = 6338,
        SDB_SIZE           = SNA3270$K_SDB_LENGTH,
        FDB_SIZE           = SNA3270$K_FDB_SIZE,
        TEXT_DSC_SIZE      = 52;

                        .
                        .
                        .

OWN
        ACCESS_DSC      :BLOCK [8, BYTE],
        DATA_DSC        :BLOCK [8, BYTE],
        NODE_DSC        :BLOCK [8, BYTE],
        CHAR_DSC        :BLOCK [8, BYTE],
        ATTR_DSC        :BLOCK [8, BYTE],
        FIELD_DSC       :BLOCK [8, BYTE],
        SDB_DSC         :BLOCK [8, BYTE],
        FDB_DSC         :BLOCK [8, BYTE],
        SDB             :SNA3270_SDB,
        FDB             :SNA3270_FDB,
        CHAR_VECTOR     :VECTOR [CHAR_VECTOR_SIZE, BYTE],
        ATTR_VECTOR     :VECTOR [ATTR_VECTOR_SIZE, WORD],
        FIELD_VECTOR    :VECTOR [FIELD_VECTOR_SIZE, BYTE],
        ACCESS_BUF      :VECTOR [BUFFER_SIZE, BYTE],
        NODE_BUF        :VECTOR [BUFFER_SIZE, BYTE],
        DATA_BUFFER     :VECTOR [DATA_BUFFER_SIZE, BYTE];

BIND
        TEXT_DSC = %ASCID %STRING (
         'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz')
                        :BLOCK [8, BYTE];

        STATUS_DSC [DSC$W_LENGTH]  = STATUS_VECTOR_SIZE;
        STATUS_DSC [DSC$A_POINTER] = STATUS_VECTOR;
        STATUS_DSC [DSC$B_CLASS]   = DSC$K_CLASS_S;

        NOTIFY_DSC [DSC$W_LENGTH]  = NOTIFY_VECTOR_SIZE;
        NOTIFY_DSC [DSC$A_POINTER] = NOTIFY_VECTOR;
        NOTIFY_DSC [DSC$B_CLASS]   = DSC$K_CLASS_S;

        FIELD_DSC [DSC$W_LENGTH]  = FIELD_VECTOR_SIZE;
        FIELD_DSC [DSC$A_POINTER] = FIELD_VECTOR;
        FIELD_DSC [DSC$B_CLASS]   = DSC$K_CLASS_S;

                        .
                        .
                        .
        STATUS = SNA3270$REQUEST_CONNECT_W (
                                  SESS_ID,
                                  STATUS_DSC,
                                  %REF (SNA3270$K_ACTIVE),
                                  %REF (SNA3270$K_FIELD_MODE),
                                  NODE_DSC,
                                  ACCESS_DSC,
                              [2] 0,0,0,0,0,0,0,
                                  CHAR_DSC,
                                  ATTR_DSC,
                                  FIELD_DSC,
                                  SDB_DSC,
                                  FDB_DSC,
                                  NOTIFY$RTN,
                                  0,
                                  NOTIFY_DSC,
                                  0,0,0,
                                  );

        IF NOT .STATUS
          THEN
             $PUTMSG (MSGVEC = STATUS_VECTOR);
!
!
! Receive the CICS logo
!
!
        STATUS = SNA3270$RECEIVE_SCREEN_W (
                                            SESS_ID,
                                            STATUS_DSC);

        IF NOT .STATUS
          THEN
             $PUTMSG (MSGVEC = STATUS_VECTOR);

                        .
                    [3] .
                        .
!
!
! Transmit CSFE
!
!
        DATA_BUFFER [0] = %X '83';
        DATA_BUFFER [1] = %X 'A2';
    [4] DATA_BUFFER [2] = %X '86';
        DATA_BUFFER [3] = %X '85';

        DATA_DSC [DSC$W_LENGTH] = 4;
        DATA_DSC [DSC$A_POINTER] = DATA_BUFFER;
        STATUS = SNA3270$WRITE_FIELD (
                                   SESS_ID,
                                   STATUS_DSC,
                                   DATA_DSC);

        STATUS = SNA3270$TRANSMIT_SCREEN_W (
                                  SESS_ID,
                                  STATUS_DSC,
                                  %REF (SNA3270$K_AID_ENTER));

        IF NOT .STATUS
          THEN
             $PUTMSG (MSGVEC = STATUS_VECTOR);
!
!
! Receive CSFE screen
!
!
        STATUS = SNA3270$RECEIVE_SCREEN_W (
                                            SESS_ID,
                                            STATUS_DSC);

        IF NOT .STATUS
          THEN
             $PUTMSG (MSGVEC = STATUS_VECTOR);

        DATA_DSC [DSC$W_LENGTH]  = TEXT_DSC_SIZE;
        DATA_DSC [DSC$A_POINTER] = DATA_BUFFER;

        STATUS = LIB$TRA_ASC_EBC (TEXT_DSC, DATA_DSC);
!
!
! Write contents of buffer in the field and transmit
!
!
    [5] STATUS = SNA3270$WRITE_FIELD (
                                   SESS_ID,
                                   STATUS_DSC,
                                   DATA_DSC);

        STATUS = SNA3270$TRANSMIT_SCREEN_W (
                                      SESS_ID,
                                      STATUS_DSC,
                                      %REF (SNA3270$K_AID_ENTER));
                        .
                        .
                        .
        STATUS = SNA3270$REQUEST_DISCONNECT_W (
                                            SESS_ID,
                                            STATUS_DSC);
        IF NOT .STATUS
          THEN
             $PUTMSG (MSGVEC = STATUS_VECTOR);

    END;
GLOBAL ROUTINE NOTIFY$RTN (EVENT_CODE_PTR, EVENT_PARM_PTR)
                                                  : NOVALUE =
BEGIN

BIND
        EVENT_CODE = .EVENT_CODE_PTR,
        EVENT_PARM = .EVENT_PARM_PTR;

LOCAL
        STATUS_VECTOR   : VECTOR [SNA3270$K_MIN_STATUS_VECTOR, LONG],
        STATUS_DSC      : BLOCK [8, BYTE],
        STATUS;

        STATUS = LIB$PUT_OUTPUT ($DESCRIPTOR (’ ’));
        STATUS = LIB$PUT_OUTPUT ($DESCRIPTOR
                                (’ Entering Notify routine’));

        $PUTMSG (MSGVEC = NOTIFY_VECTOR);
        RETURN;
END;
END
ELUDOM
```

Comments

1. Include the 3270 Library.
2. The zeros and commas indicate that you do not want to specify values for the parameters and will accept the default values provided by the Interface.
3. To write the screen image, you must clear the screen at this point.
4. The data equals "csfe".
5. Note that the FDB is not manipulated. The screen image is unformatted, unlike in the FORTRAN example (Section 5.1).

5.6 MACRO Programming Example–Data Stream Mode

This MACRO fragment initiates a session with CICS using stream mode. It then invokes the CSFE transaction (a remote loopback program) running under CICS and transmits a string of data to it. After receiving the CSFE screen from IBM, the program terminates the session.

```
           .TITLE MARSM
           SNA3270DF

.PSECT  RWDATA,WRT,NOEXE,QUAD

[1] PROMPT:  .ASCID  /Entering notify routine /
  STATUS : .BLKL   10
  SESS_ID: .LONG   0                               ;session-id
  STS_VEC: .BLKB   SNA3270$K_MIN_STATUS_VECTOR     ;status-vector
  STS_DSC: .LONG   SNA3270$K_MIN_STATUS_VECTOR
           .ADDRESS STS_VEC
[2] DATA_BUF:.BLKB   1000                            ;data-buffer
[3] DATA_DSC:.LONG   1000
           .ADDRESS DATA_BUF
  DATA_LEN:.LONG   0                               ;data-length
  LAST_FLG:.LONG   0
  NT_VEC : .BLKB   SNA3270$K_MIN_NOTIFY_VECTOR     ;notify-vector
  NT_DSC:  .LONG   SNA3270$K_MIN_NOTIFY_VECTOR
           .ADDRESS NT_VEC
  ND_NAME: .ASCID  /BOOJUM/                        ;node-name
  AC_NAME: .ASCID  /XCICS/                         ;access-name
[4] TST_DATA:.ASCID /ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz/
  TMP_BUF :.BLKB   52
  TMP_DSC: .LONG   52
           .ADDRESS TMP_BUF


           .PSECT  CODE,NOWRT,EXE,LONG

       [5] .ENTRY  NOTIFY$RTN, ^M<>       ;notify-routine entry point
           PUSHAQ  PROMPT
           CALLS   #1,G^LIB$PUT_OUTPUT
           $PUTMSG_S NT_VEC               ;display notify-vector
           RET
```

```
         .ENTRY  MARSM, ^M<>            ;main program entry point
         CLRL    R0
         PUSHL   #0                     ;ast parameter
         PUSHL   #0                     ;ast address
         PUSHL   #0                     ;event-flag
         PUSHAQ  NT_DSC                 ;notify-vector
         PUSHL   #0                     ;notify-parameter
         PUSHAL  NOTIFY$RTN             ;notify-routine address
         PUSHL   #0                     ;fdb
         PUSHL   #0                     ;sdb
         PUSHL   #0                     ;field-vector
         PUSHL   #0                     ;attribute-vector
         PUSHL   #0                     ;character-vector
     [5] PUSHL   #0                     ;data
         PUSHL   #0                     ;password
         PUSHL   #0                     ;userid
         PUSHL   #0                     ;logon-mode
         PUSHL   #0                     ;application program
         PUSHL   #0                     ;session-address
         PUSHL   #0                     ;circuit
         PUSHAL  AC_NAME                ;access-name
         PUSHAL  ND_NAME                ;node-name
         PUSHAL  #SNA3270$K_STREAM_MODE ;mode-type
         PUSHAL  #SNA3270$K_ACTIVE      ;conn-type
         PUSHAQ  STS_DSC                ;status-vector
         PUSHAL  SESS_ID                ;session-id

         CALLS   #24,G^SNA3270$REQUEST_CONNECT_W
         BLBS    R0, 10$
         BRW     EXITS                  ;exit if error

10$:     PUSHL   #0                     ;ast parameter
         PUSHL   #0                     ;ast address
         PUSHL   #0                     ;event-flag
         PUSHAL  DATA_LEN               ;data-length
         PUSHAQ  DATA_DSC               ;data-buffer
         PUSHAQ  STS_DSC                ;status-vector
         PUSHAL  SESS_ID                ;session-id

         CALLS   #7,G^SNA3270$RECEIVE_STREAM_W  ;receive CICS logo
         BLBS    R0, 20$
         BRW     EXITS                  ;exit if error

20$:     PUSHAL  #SNA3270$K_ACK_ACCEPT  ;accept CICS logo
         PUSHAL  STS_DSC                ;status-vector
         PUSHAL  SESS_ID                ;session-id

         CALLS   #3,G^SNA3270$ACKNOWLEDGE
         BLBS    R0, 30$
         BRW     EXITS                  ;exit if error
```

```
30$: [6] MOVL    #^X<6D>,DATA_BUF -
                  + SNA3270$K_BUF_HDLEN ;move "clear-screen"
         MOVL    #^X<1>,LAST_FLG
         MOVL    #^X<8>,DATA_LEN

         PUSHL   #0                     ;ast-parameter
         PUSHL   #0                     ;ast-address
         PUSHL   #0                     ;event-flag
         PUSHAL  LAST_FLG               ;last-flag indicator
         PUSHAL  DATA_LEN               ;data-length
         PUSHAQ  DATA_DSC               ;data-buffer
         PUSHAQ  STS_DSC                ;status-vector
         PUSHAL  SESS_ID                ;sesion-id

         CALLS   #8,G^SNA3270$TRANSMIT_STREAM_W  ;clear screen
                                                 ;to CICS
         BLBS    R0, 40$
         BRW     EXITS                  ;exit if error

40$:     PUSHL   #0                     ;ast parameter
         PUSHL   #0                     ;ast address
         PUSHL   #0                     ;event-flag
         PUSHAL  DATA_LEN               ;data-length
         PUSHAQ  DATA_DSC               ;data-buffer
         PUSHAQ  STS_DSC                ;status-vector
         PUSHAL  SESS_ID                ;session-id

         CALLS   #7,G^SNA3270$RECEIVE_STREAM_W  ;clear screen
                                                ;command from CICS
         BLBS    R0, 50$
         BRW     EXITS                  ;exit if error

50$:     PUSHAL  #SNA3270$K_ACK_ACCEPT
         PUSHAL  STS_DSC                ;status-vector
         PUSHAL  SESS_ID                ;session-id

         CALLS   #3,G^SNA3270$ACKNOWLEDGE
         BLBS    R0, 60$
         BRW     EXITS                  ;exit if error

60$: [7] MOVQ    #^X<8586A283C4407D>,DATA_BUF -
                 + SNA3270$K_BUF_HDLEN ;move "csfe"

         MOVL    #^X<1>,LAST_FLG
         MOVL    #^X<E>,DATA_LEN
                .
                .
                .
         PUSHL   #0                     ;ast-parameter
90$:     PUSHAQ  TMP_DSC                ;temporary buffer
         PUSHAQ  TST_DATA               ;test-data
```

```
     [8] CALLS   #2,G^LIB$TRA_ASC_EBC   ;translate data to ebcdic
         MOVL    #^X<1>,LAST_FLG
         MOVL    #^D<62>,DATA_LEN
         MOVL    #^D<52>,R3             ;set maximum index value
         MOVL    #^X<0>,R4              ;initialize index

         MOVL    #^X<C4407D>,DATA_BUF -
                  + SNA3270$K_BUF_HDLEN ;move control data after header

     [9] MOVL    #^X<A>,R5              ;initialize index leaving
                                        ;10 bytes for header/control data

LOOP:    MOVB    TMP_BUF[R4] ,DATA_BUF[R5]
         ADDL    I^#1,R4                ;transfer data from temp buffer
         ADDL    I^#1,R5                ;to data-buffer before
         CMPL    R4, R3                 ;transmitting
         BNEQ    LOOP

         PUSHL   #0                     ;ast-parameter
         PUSHL   #0                     ;ast-address
         PUSHL   #0                     ;event-flag
         PUSHAL  LAST_FLG               ;last-flag indicator
         PUSHAL  DATA_LEN               ;data-length
         PUSHAQ  DATA_DSC               ;data-buffer
         PUSHAQ  STS_DSC                ;status-vector
         PUSHAL  SESS_ID                ;sesion-id

         CALLS   #8,G^SNA3270$TRANSMIT_STREAM_W
                        .
                        .
                        .
110$:    PUSHL   #0                     ;ast-parameter
         PUSHL   #0                     ;ast-address
         PUSHL   #0                     ;event-flag
         PUSHAQ  STSD_SC                ;status-vector
         PUSHAL  SESS_ID                ;session-id

         CALLS   #5,G^SNA3270$REQUEST_DISCONNECT_W

EXITS:[10] $PUTMSG_S STS_VEC              ;display status-vector
         $EXIT_S
         .END   MARSM
```

Comments

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

   ```
   $ MACRO/OBJECT=MYDIR:MYPROG SYS$LIBRARY:SNA3270DF+MYDIR:MYPROG
   ```

   where

   MYDIR and MYPROG are your directory and program.

2. Note that the program uses a null class and type; that is, the class and type byte = 0.
3. Test string being sent to IBM.
4. Note that in this program the notify routine only indicates that the application received the event. Normally, the application would take some action.
5. Arguments are passed on the stack.
6. The 6D represents an AID key code (clear) that is positioned after the Interface header.
7. This data represents three pieces of information. Note the reverse order of the data; it is read starting from the least significant byte.
   - 8586A283 = CSFE
   - C440 = cursor address
   - 7D = an AID key (enter)
8. You can use OpenVMS Library routines to do parts of your application, such as translating ASCII to EBCDIC or vice versa.
9. Note the reverse order of the data; it is read starting from the least significant byte.
10. Display the status vector by using $PUTMSG.

5.7 Pascal Programming Example–Data Stream Mode

This Pascal program fragment initiates a session with CICS using stream mode. It then invokes the CSFE transaction (a remote loopback program) running under CICS and transmits a string of data to it, then terminates the session.

```
[1] [INHERIT('SNA3270DF.PEN')] PROGRAM TESTPAS(INPUT,OUTPUT);

  LABEL
          10;

  CONST
          BUFFER_SIZE = 132;

  VAR
          SESSION_ID   :INTEGER;
          CONN_TYPE    :INTEGER;
          MODE_TYPE    :INTEGER;
          NOTIFY_VECTOR:PACKED ARRAY [1..SNA3270$K_MIN_NOTIFY_VECTOR]
                                                             OF CHAR;
          STATUS_VECTOR:PACKED ARRAY [1..SNA3270$K_MIN_NOTIFY_VECTOR]
                                                             OF CHAR;
          NODE_NAME    :PACKED ARRAY [1..6]  OF CHAR;
          ACCESS_NAME  :PACKED ARRAY [1..8]  OF CHAR;
          STATUS       :INTEGER;
          DATA_LENGTH  :INTEGER;
          IDX          :INTEGER;
          TMP_IDX      :INTEGER;
          OUT_BUFFER   :PACKED ARRAY[1..52]   OF CHAR;
          TEMP_BUFFER  :PACKED ARRAY[1..52]   OF CHAR;
          DATA_BUFFER  :PACKED ARRAY[1..1000] OF CHAR;


  [EXTERNAL,ASYNCHRONOUS] FUNCTION LIB$TRA_ASC_EBC
      (%STDESCR TEMP_BUFFER:PACKED ARRAY[$l1..$u1:INTEGER] OF CHAR;
       %STDESCR OUT_BUFFER :PACKED ARRAY[$l2..$u2:INTEGER] OF CHAR)
                                                 :INTEGER; EXTERNAL;

  [ASYNCHRONOUS,EXTERNAL(SYS$PUTMSG)] FUNCTION $PUTMSG
                  .
                  .
                  .

  PROCEDURE NOTIFY_RTN;

  BEGIN

  WRITELN;
  WRITELN  ('Entering notify routine');

  END;

  BEGIN   (* Main program *)
```

```
(***************************************************************)
(* Get node and access names                                   *)
(***************************************************************)
                .
                .
                .
(***************************************************************)
(* Request connect                                             *)
(***************************************************************)

STATUS := SNA3270$REQUEST_CONNECT_W (SESSION_ID,STATUS_VECTOR,
                                     SNA3270$K_ACTIVE,
                                     SNA3270$K_STREAM_MODE,
                                     NODE_NAME,
                                     ACCESS_NAME,
                                 [2] ,,,,,,,,,,,,
                                     %IMMED NOTIFY_RTN,,
                                     NOTIFY_VECTOR);

IF NOT (STATUS) :: BOOLEAN
  THEN
     GOTO 10;

(***************************************************************)
(* Receive CICS logo                                           *)
(***************************************************************)

STATUS := SNA3270$RECEIVE_STREAM_W (SESSION_ID,
                                    STATUS_VECTOR,
                                    DATA_BUFFER,
                                    DATA_LENGTH);

IF NOT (STATUS) :: BOOLEAN
                .
                .
                .
(***************************************************************)
(* Acknowledge CICS logo                                       *)
(***************************************************************)

STATUS := SNA3270$ACKNOWLEDGE (SESSION_ID,
                               STATUS_VECTOR,
                               SNA3270$K_ACK_ACCEPT);
                .
                .
                .
  (***************************************************************)
  (* Transmit CSFE                                               *)
  (***************************************************************)

  DATA_BUFFER [SNA3270$K_BUF_HDLEN + 1] :=’’(%X’7D’)’’; (* cont- *)
  DATA_BUFFER [SNA3270$K_BUF_HDLEN + 2] :=’’(%X’40’)’’; (* rol   *)
  DATA_BUFFER [SNA3270$K_BUF_HDLEN + 3] :=’’(%X’C4’)’’; (* data  *)
[3] DATA_BUFFER [SNA3270$K_BUF_HDLEN + 4] :=’’(%X’83’)’’; (* ’c’ *)
  DATA_BUFFER [SNA3270$K_BUF_HDLEN + 5] :=’’(%X’A2’)’’; (* ’s’ *)
  DATA_BUFFER [SNA3270$K_BUF_HDLEN + 6] :=’’(%X’86’)’’; (* ’f’ *)
  DATA_BUFFER [SNA3270$K_BUF_HDLEN + 7] :=’’(%X’85’)’’; (* ’e’ *)

  DATA_LENGTH    := 14;

  STATUS := SNA3270$TRANSMIT_STREAM_W (SESSION_ID,
                                       STATUS_VECTOR,
                                       DATA_BUFFER,
                                       DATA_LENGTH,
                                       SNA3270$K_END_OF_DATA);
                  .
                  .
                  .
  (***************************************************************)
  (* Receive CSFE                                                *)
  (***************************************************************)

  STATUS := SNA3270$RECEIVE_STREAM_W (SESSION_ID,
                                      STATUS_VECTOR,
                                      DATA_BUFFER,
                                      DATA_LENGTH);
                  .
                  .
                  .
  (***************************************************************)
  (* Acknowledge CSFE                                            *)
  (***************************************************************)

  STATUS := SNA3270$ACKNOWLEDGE (SESSION_ID,
                                 STATUS_VECTOR,
                                 SNA3270$K_ACK_ACCEPT);
                  .
                  .
                  .
  (***************************************************************)
  (* Translate data from ASCII to EBCDIC                         *)
  (***************************************************************)

[4] TEMP_BUFFER :=
            ’ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz’;

[5] STATUS := LIB$TRA_ASC_EBC (TEMP_BUFFER,OUT_BUFFER);
```

```
  (***************************************************************)
  (* Move control information and data to data-buffer after      *)
  (* header                                                      *)
  (***************************************************************)

  DATA_BUFFER [SNA3270$K_BUF_HDLEN + 1] :=’’(%X’7D’)’’;
[6] DATA_BUFFER [SNA3270$K_BUF_HDLEN + 2] :=’’(%X’40’)’’;
  DATA_BUFFER [SNA3270$K_BUF_HDLEN + 3] :=’’(%X’C4’)’’;

  IDX     := 11;
  TMP_IDX := 1;

  WHILE TMP_IDX < 53 DO
     BEGIN
         DATA_BUFFER[IDX] := OUT_BUFFER[TMP_IDX];

         IDX     := IDX + 1;
         TMP_IDX := TMP_IDX + 1;
     END;

  DATA_LENGTH    := 62;

  (***************************************************************)
  (* Transmit data                                               *)
  (***************************************************************)

  STATUS := SNA3270$TRANSMIT_STREAM_W (SESSION_ID,
                                       STATUS_VECTOR,
                                       DATA_BUFFER,
                                       DATA_LENGTH,
                                       SNA3270$K_END_OF_DATA);

                  .
                  .
                  .

  (***************************************************************)
  (* Disconnect link                                             *)
  (***************************************************************)

  STATUS := SNA3270$REQUEST_DISCONNECT_W (SESSION_ID,
                                          STATUS_VECTOR);

[7] 10: $PUTMSG (STATUS_VECTOR);

  END.
```

Comments

1. The symbol definition file, SNA3270$DF.PAS, has been precompiled into an environment file. The MODULE and END statements must be deleted from the file in order to use it as an include file.
2. The commas indicate that you do not want to specify values for the parameters and will accept the default values provided by the Interface.
3. This data represents three pieces of information:
   - 7D = an AID key (enter)
   - 40C4 = cursor address
   - 83A28685 = CSFE
4. Test string is being sent to IBM.
5. You can use OpenVMS Library routines to do parts of your application, such as translating ASCII to EBCDIC or vice versa.
6. This data represents two pieces of information:
   - 7D = an AID key (enter)
   - 40C4 = cursor address
7. Display the status vector by using $PUTMSG.

Part III

Reference

6 Procedure Calling Formats

This chapter describes the calling formats for the procedures provided by the Digital SNA 3270 Data Stream Programming Interface. These procedures include:

- SNA3270$ACKNOWLEDGE
- SNA3270$LOCK_SCREEN
- SNA3270$READ_FIELD
- SNA3270$RECEIVE_SCREEN
- SNA3270$RECEIVE_STREAM
- SNA3270$REQUEST_CONNECT
- SNA3270$REQUEST_DISCONNECT
- SNA3270$TRANSMIT_LUSTAT
- SNA3270$TRANSMIT_SCREEN
- SNA3270$TRANSMIT_SIGNAL
- SNA3270$TRANSMIT_STREAM
- SNA3270$WRITE_FIELD

Calls to the 3270 Interface procedures have the following general format:

status=SNA3270$procedure-name[_W](argument,...,)[argument]

where

status	is a status code returned as a function value.
procedure-name	is the name of the Interface procedure that you want to call.
_W	specifies a synchronous operation.
( )	delimits the argument list.

[argument] indicates an optional argument.

argument is a variable containing information that the application passes to or receives from the Interface. 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 B.

You can pass arguments to the Interface 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 descriptor.

In this chapter, the shorthand notation for each procedure specify how each argument is to be passed. For more information see the *OpenVMS RTL Library (LIB$) Manual*.

6.1 SNA3270$ACKNOWLEDGE

The SNA3270$ACKNOWLEDGE procedure informs the IBM application subsystem whether the data it received by means of the SNA3270$RECEIVE_STREAM procedure is acceptable. This procedure is only used in data stream mode. The Interface performs all acknowledgment in field mode. The SNA3270$ACKNOWLEDGE always returns synchronously.

Format:

status.wlc.v=SNA3270$ACKNOWLEDGE (*session-id.rlu.r*,
status-vec.wz.dx,
sense-code.rlu.r)

Arguments:

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

session-id
The session identifier assigned at connect time. Passed by reference.

status-vec
A longword vector allocated by the OpenVMS application and filled in by the Interface to provide the user with complete status information. Passed by descriptor.

sense-code
A longword containing the response to be sent to the PLU. Passed by reference. The 3270 documentation offers a multitude of possible values for this parameter. Some of the most common are listed as follows and are defined symbolically in the definition file (see Appendix C):

SNA3270$K_ACK_ACCEPT—Accept
SNA3270$K_ACK_NOFUNC—Function not supported
SNA3270$K_ACK_PRMERR—Parameter error
SNA3270$K_ACK_NOCATG—Category not supported
SNA3270$K_ACK_INTREQ—Intervention required
SNA3270$K_ACK_NOPROC—Procedure not supported
SNA3270$K_ACK_PSILOS—Presentation space integrity lost
SNA3270$K_ACK_SLUBUS—SLU busy

The SNA3270$ACKNOWLEDGE procedure can return the following status messages.

- SNA3270$_OK
- SNA3270$_ACKFAI
- SNA3270$_INVSID
- SNA3270$_NORSPPEND

When you write an application for the 3270 Data Stream Programming Interface, the application needs to check for the different status messages that the Interface can return to the application. For example, in some procedures, if the application receives the SNA3270$_NYTXMIT (not your turn to transmit), it cannot issue a SNA3270$TRANSMIT_STREAM procedure. If the application checks only for an SNA3270$_OK status at this point, it will miss the SNA3270$_NYTXMIT and it may try to issue a SNA3270$TRANSMIT_STREAM. If the application tries to issue a SNA3270$TRANSMIT_STREAM at this time, you may receive status messages from the Interface that you do not expect. Therefore, you need to check for the particular status code returned and not just for the SNA3270$_OK status. For further information about the success status messages, refer to Section D.1.

6.2 SNA3270$LOCK_SCREEN

The SNA3270$LOCK_SCREEN procedure causes the Interface to send a negative response to any BID request received from the IBM host. The lock screen condition is removed by a subsequent transmit stream or screen request. This procedure always completes synchronously. See Section 2.4.4 for more information.

Format:

status.wlc.v=SNA3270$LOCK_SCREEN (*session-id.rlu.r*, *status-vec.wz.dx*)

Arguments:

status	When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits together represent the severity of the error. Returned as a function value.
session-id	The session identifier assigned at connect time. Passed by reference.
status-vec	A longword vector allocated by the OpenVMS application and filled in by the Interface to provide the user with complete status information. Passed by descriptor.

The SNA3270$LOCK_SCREEN procedure can return the following status messages.

- SNA3270$_OK
- SNA3270$_INVSID
- SNA3270$_LOCFAI

When you write an application for the 3270 Data Stream Programming Interface, the application needs to check for the different status messages that the Interface can return to the application. For example, in some procedures, if the application receives the SNA3270$_NYTXMIT (not your turn to transmit), it cannot issue a SNA3270$TRANSMIT_STREAM procedure. If the application checks only for an SNA3270$_OK status at this point, it will miss the SNA3270$_NYTXMIT and it may try to issue a SNA3270$TRANSMIT_STREAM. If the application tries to issue a SNA3270$TRANSMIT_STREAM at this time, you may receive status messages from the Interface that you do not expect. Therefore, you need to check for the particular status code returned and not just for the SNA3270$_OK status. For further information about the success status messages, refer to Section D.1.

6.3 SNA3270$READ_FIELD

The SNA3270$READ_FIELD procedure reads the specified field from the display vectors. You describe the desired field characteristics in the field descriptor block (FDB). See Section 2.4.5 for information about the use of the FDB. This procedure always completes synchronously.

Format:

```
                                    (session-id.rlu.r,
status.wlc.v=SNA3270$READ_FIELD     status-vec.wz.dx,
                                    [data-bufr.wr.dx])
```

Arguments:

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

session-id The session identifier assigned at connect time. Passed by reference.

status-vec A longword vector allocated by the OpenVMS application and filled in by the Interface to provide the user with complete status information. Passed by descriptor.

data-bufr The descriptor of a buffer to receive the field specified in the field descriptor block. If null, no data is copied into the buffer and the Interface only returns the field offset and size of the field in the FDB. Passed by descriptor.

The SNA3270$READ_FIELD procedure can return the following status messages.

- SNA3270$_OK
- SNA3270$_OK_TRUNC
- SNA3270$_BADOFFSET
- SNA3270$_INVSID
- SNA3270$_NOFIELD
- SNA3270$_NOTINFMOD
- SNA3270$_RDFLDFAI

When you write an application for the 3270 Data Stream Programming Interface, the application needs to check for the different status messages that the Interface can return to the application. For example, in some procedures, if the application receives the SNA3270$_NYTXMIT (not your turn to transmit), it cannot issue a SNA3270$TRANSMIT_STREAM procedure. If the application checks only for an SNA3270$_OK status at this point, it will miss the SNA3270$_NYTXMIT and it may try to issue a SNA3270$TRANSMIT_ STREAM. If the application tries to issue a SNA3270$TRANSMIT_STREAM at this time, you may receive status messages from the Interface that you do not expect. Therefore, you need to check for the particular status code returned and not just for the SNA3270$_OK status. For further information about the success status messages, refer to Section D.1.

6.4 SNA3270$RECEIVE_SCREEN

The SNA3270$RECEIVE_SCREEN procedure receives a complete 3270 screen image and places it in the display vectors specified for the session. This procedure updates the screen descriptor block (SBD). See Section 2.4.3 for more information about the SDB.

Format:

status.wlc.v=SNA3270$RECEIVE_SCREEN[_W] (*session-id.rlu.r*,
status-vec.w.dx,
[*event-flag.rlu.r*],
[*ast-addr.szem.r*],
[*ast-par.rlu.r*])

Arguments:

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

session-id The session identifier assigned at connect time. Passed by reference.

status-vec A longword vector allocated by the OpenVMS application and filled in by the Interface to provide the user with complete status information. Passed by descriptor.

event-flag An event flag to be set at completion. Passed by reference.

ast-addr A user-written procedure called by the application 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 SNA3270$RECEIVE_SCREEN procedure can return the following status messages.

- SNA3270$_OK
- SNA3270$_OK_CONT
- SNA3270$_OK_NYT
- SNA3270$_INVSID
- SNA3270$_NOTINFMOD

- SNA3270$_NYTRCV
- SNA3270$_SCRACT

When you write an application for the 3270 Data Stream Programming Interface, the application needs to check for the different status messages that the Interface can return to the application. For example, in some procedures, if the application receives the SNA3270$_NYTXMIT (not your turn to transmit), it cannot issue a SNA3270$TRANSMIT_STREAM procedure. If the application checks only for an SNA3270$_OK status at this point, it will miss the SNA3270$_NYTXMIT and it may try to issue a SNA3270$TRANSMIT_STREAM. If the application tries to issue a SNA3270$TRANSMIT_STREAM at this time, you may receive status messages from the Interface that you do not expect. Therefore, you need to check for the particular status code returned and not just for the SNA3270$_OK status. For further information about the success status messages, refer to Section D.1.

6.5 SNA3270$RECEIVE_STREAM

The SNA3270$RECEIVE_STREAM procedure receives a 3270 data stream transmitted from the IBM host.

Format:

```
                                               (session-id.rlu.r,
                                               status-vec.wz.dx,
                                               [data-bufr.wr.dx],
status.wlc.v=SNA3270$RECEIVE_STREAM[_W]  [data-length.wlu.r],
                                               [event-flag.rlu.r],
                                               [ast-addr.zem.r],
                                               [ast-par.rlu.r])
```

Arguments:

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

session-id The session identifier assigned at connect time. Passed by reference.

status-vec A longword vector allocated by the OpenVMS application and filled in by the Interface to provide the user with complete status information. Passed by descriptor.

data-bufr The data buffer to receive the 3270 data stream. Passed by descriptor.

data-length A longword variable to receive the number of bytes sent by IBM. Passed by reference.

event-flag An event flag to be set at completion. Passed by reference.

ast-addr A user-written procedure called by the application 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 SNA3270$RECEIVE_STREAM procedure can return the following status messages.

- SNA3270$_OK
- SNA3270$_OK_CONT
- SNA3270$_OK_MORE
- SNA3270$_OK_NYT
- SNA3270$_BUFSMALL
- SNA3270$_INVSID

- SNA3270$_NYTRCV
- SNA3270$_RECVSTFAI

When you write an application for the 3270 Data Stream Programming Interface, the application needs to check for the different status messages that the Interface can return to the application. For example, in some procedures, if the application receives the SNA3270$_NYTXMIT (not your turn to transmit), it cannot issue a SNA3270$TRANSMIT_STREAM procedure. If the application checks only for an SNA3270$_OK status at this point, it will miss the SNA3270$_NYTXMIT and it may try to issue a SNA3270$TRANSMIT_STREAM. If the application tries to issue a SNA3270$TRANSMIT_STREAM at this time, you may receive status messages from the Interface that you do not expect. Therefore, you need to check for the particular status code returned and not just for the SNA3270$_OK status. For further information about the success status messages, refer to Section D.1.

6.6 SNA3270$REQUEST_CONNECT

The SNA3270$REQUEST_CONNECT procedure issues an active or passive request to establish a data stream mode or field mode session between an OpenVMS application and IBM application.

Format:

```
                                         (session-id.wlu.r,
                                         status-vec.wz.dx,
                                         conn-typ.rlu.r,
                                         mode-typ.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],
status.wlc.v=SNA3270$REQUEST_CONNECT[_W] [data.rt.dx],
                                         [char-vec.mx.dx],
                                         [attr-vec.mx.dx],
                                         [field-vec.mx.dx],
                                         [sdb-dsc.wz.dx],
                                         [fdb-dsc.mz.dx],
                                         [notify-rtn.zem.r],
                                         [notify-parm.rlu.r],
                                         [notify-vec.wz.dx],
                                         [event-flag.rlu.r],
                                         [ast-addr.zem.r],
                                         [ast-par.rlu.r]
                                         [lu-password.rt.dx]
                                         [pid.rlu.r])
```

Arguments:

status — When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits together represent the severity of the error. Returned 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-vec
A longword vector allocated by the OpenVMS application and filled in by the Interface to provide the user with complete status information. Passed by descriptor.

conn-typ
A value that specifies the type of connection desired. SNA3270$K_ACTIVE indicates an active connection. SNA3270$K_PASSIVE indicates a passive connection. Passed by reference.

mode-typ
A value that specifies the mode of connection desired. The connection is either a SNA3270$K_STREAM_MODE connection or a SNA3270$K_FIELD_MODE connection. Passed by reference.

node-desc
A Gateway DECNet node name or TCP/IP host name string. **For OpenVMS SNA, set this parameter equal to an ASCII 0.** If this parameter is not supplied, the Interface assumes you are requesting a connection by means of OpenVMS SNA. Passed by descriptor.

acc-name
An access name associated with a list of default PLU access values. The maximum length is 8 bytes. If you omit the access name, you must supply some or all of the required IBM access information in the six parameters that follow. Passed by descriptor.

pu-name
A string defining the Gateway PU or LU used to establish the session with IBM. The maximum length is 8 bytes. For OpenVMS SNA, DECnet SNA Gateway-CT and DECnet SNA Gateway-ST, this parameter contains a PU name in the form of SNA-#, where # is a value between 0 and 3.

For the Digital SNA Domain Gateway and Digital SNA Peer Server, this parameter contains an LU name as defined in the Gateway. If this parameter is not supplied or specifies a zero length descriptor, the appropriate information is taken from the access name.

Passed by descriptor.

sess-addr
The number of the SLU over which the session is to take place. Passed by reference.

applic-prog
A string defining the PLU application that you want to connect to in the IBM host. The maximum length is 8 bytes. Note that most IBM application names must be uppercase (for example, CICS). Passed by descriptor.

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

user-id
A string identifying the user to the SSCP. The maximum length is 8 bytes. Passed by descriptor.

pass-word
A string specifying the password associated with the user ID. The maximum length is 8 bytes. Passed by descriptor.

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

char-vec

The character vector is required for field mode connection only. The MODE_TYP must be SNA3270$K_FIELD_MODE. The character vector must be one byte greater than the larger of the default or alternate display size specified in the connection parameters received from IBM. If it is smaller than what IBM specified, the request completes with an error. Each character in the display is represented by a byte. Passed by descriptor.

attr-vec

The attribute vector is required for field mode connection only. The MODE_TYP must be SNA3270$K_FIELD_MODE. Each character in the display is represented by a word in the attribute vector. Passed by descriptor.

field-vec

The field vector is required for field mode connection only. The MODE_TYP must be SNA3270$K_FIELD_MODE. Each character in the display is represented by a bit in the field vector. If the bit is set (1), then this character position marks the start of the field. All other bits are set off (0). Passed by descriptor.

sdb-dsc

The screen descriptor block is required for field mode connection only. It is a data structure used to describe the screen image (see Figure 2–4). The write control character, cursor address, and screen format are contained in this structure. Passed by descriptor.

fdb-dsc

The field descriptor block is required for field mode connection only. It is a data block used to describe a field in the screen image (see Figure 2–5). The OpenVMS application uses the field descriptor block to supply the address or attributes, or both, of the field it wants the Interface to read with the SNA3270$READ_FIELD procedure. Upon successful completion of the read procedure, the Interface uses the FDB to supply the address and attributes of the field it has placed in the user's buffer. Passed by descriptor.

notify-rtn

The address of the notification procedure. This procedure is called by the Interface to notify the user application of network-related events. Passed by reference.

notify-parm

An optional user-specified parameter to be passed to the notification procedure. Passed by reference.

notify-vec

A longword vector allocated by the OpenVMS application and filled with asynchronous event information by the Interface. The application may display the event via the system service call $PUTMSG. This structure should be global so the notify routine can reference it. Passed by descriptor.

event-flag

An event flag to be set upon completion. Passed by reference.

ast-addr

A user-written procedure called by the application 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.

lu-password A DEC multinational character string used to supply an authorization password that may be required for access to a particular LU. Passed by descriptor.

pid The OpenVMS identification of the process on whose behalf the connection is being made. The caller needs GROUP or WORLD privilege to specify a process that does not have the same UIC as the calling process.

The 3270DS interface passes a username and terminal name to the Gateway which may be needed for LU authorization purposes. Ordinarily, the interface passes the username and terminal for the process in which it is running. However, you can change the username and terminal indirectly by specifying the "pid" parameter. The pid is used when the process issuing the call is doing so on behalf of some other process in the system.

The SNA3270$REQUEST_CONNECT procedure can return the following status messages.

- SNA3270$_OK
- SNA3270$_ATTRSHO
- SNA3270$_BADVEC
- SNA3270$_CHARSHO
- SNA3270$_CONFAI
- SNA3270$_FDBLENERR
- SNA3270$_FVECSHO
- SNA3270$_ILLCONTYP
- SNA3270$_SCRLENERR
- SNA3270$_UNABD0
- SNA3270$_UNABD1
- SNA3270$_UNABD2
- SNA3270$_UNARANGE
- SNA3270$_UNAVALUE

When you write an application for the 3270 Data Stream Programming Interface, the application needs to check for the different status messages that the Interface can return to the application. For example, in some procedures, if the application receives the SNA3270$_NYTXMIT (not your turn to transmit), it cannot issue a SNA3270$TRANSMIT_STREAM procedure. If the application checks only for an SNA3270$_OK status at this point, it will miss the SNA3270$_NYTXMIT and it may try to issue a SNA3270$TRANSMIT_STREAM. If the application tries to issue a SNA3270$TRANSMIT_STREAM at this time, you may receive status messages from the Interface that you do not expect. Therefore, you need to check for the particular status code returned and not just for the SNA3270$_OK status. For further information about the success status messages, refer to Section D.1.

6.7 SNA3270$REQUEST_DISCONNECT

The SNA3270$REQUEST_DISCONNECT procedure initiates immediate termination of the session. The Interface disconnects from the IBM network and deallocates session resources. SNA3270$REQUEST_DISCONNECT must be called for any session that is started even if the session has terminated due to an asynchronous event.

Format:

status.wlc.v=SNA3270$REQUEST_DISCONNECT[_W] (*session-id.rlu.r*,
status-vec.wz.dx,
[*event-flag.rlu.r*],
[*ast-addr.zem.r*],
[*ast-par.rlu.r*])

Arguments:

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

session-id
The session identifier assigned at connect time. Passed by reference.

status-vec
A longword vector allocated by the OpenVMS application and filled in by the Interface to provide the user with complete status information. Passed by descriptor.

event-flag
An event flag to be set at completion. Passed by reference.

ast-addr
A user-written procedure called by the application 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 SNA3270$REQUEST_DISCONNECT procedure can return the following status messages.

- SNA3270$_OK
- SNA3270$_DISFAI
- SNA3270$_INVSID

When you write an application for the 3270 Data Stream Programming Interface, the application needs to check for the different status messages that the Interface can return to the application. For example, in some procedures, if the application receives the SNA3270$_NYTXMIT (not your turn to transmit), it cannot issue a SNA3270$TRANSMIT_STREAM procedure. If the application checks only for an SNA3270$_OK status at this point, it will miss the SNA3270$_NYTXMIT and it may try to issue a SNA3270$TRANSMIT_STREAM. If the application tries to issue a SNA3270$TRANSMIT_STREAM at this time, you may receive status messages from the Interface that you do not expect. Therefore, you need to check for the particular status code returned and not just for the SNA3270$_OK status. For further information about the success status messages, refer to Section D.1.

6.8 SNA3270$TRANSMIT_LUSTAT

The SNA3270$TRANSMIT_LUSTAT procedure sends a Logical Unit Status (LUSTAT) RU, to the LU on the IBM host. In general, the LUSTAT is used to report failures and error recovery conditions for a local device of an LU. The LUSTAT status value and status extension field, each 2 bytes in length, if specified by the caller, is defined in the optional *data-bufr* parameter. The LUSTAT status value and status extension field is a four byte status field as described the *IBM Systems Network Architecture Formats* manual. If the *data-bufr* parameter is omitted, the default is a status value of X'082B' and a status extention field of 0.

Format:

status.wlc.v=SNA3270$TRANSMIT_LUTSTAT (*session-id.wlu.r*,
status-vec.wz.dx,
[*data-bufr.rt.dx*])

Arguments:

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

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

status-vec — A longword vector allocated by the OpenVMS application and filled in by the Interface to provide the user with complete status information. Passed by descriptor.

data-bufr — A maximum 4 byte data buffer containing the status value and status extention field of the LUSTAT RU. If omitted the default is a status value of X'082B' and a status extention field of 0. Passed by descriptor.

The SNA3270$TRANSMIT_LUSTAT procedure can return the following status messages.

- SNA3270$_OK
- SNA3270$_XMTLUSTFAI
- SNA3270$_FUNCABORT
- SNA3270$_XMITSTFAI
- SNA3270$_DFCERR
- SNA3270$_INVSID

6.9 SNA3270$TRANSMIT_SCREEN

The SNA3270$TRANSMIT_SCREEN procedure interprets the character and attribute vectors to generate a 3270 data stream and transmits the complete 3270 data stream to the IBM host.

Format:

status.wlc.v=SNA3270$TRANSMIT_SCREEN[_W] (*session-id.rlu.r*,
status-vec.wz.dx,
aid.rw.r,
[*event-flag.rlu.r*],
[*ast-addr.zem.r*],
[*ast-par.rlu.r*])

Arguments:

status	When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits together represent the severity of the error. Returned as a function value.
session-id	The session identifier assigned at connect time. Passed by reference.
status-vec	A longword vector allocated by the OpenVMS application and filled in by the Interface to provide the user with complete status information. Passed by descriptor.
aid	The attention identification (AID) is a code that the OpenVMS application sends to the host, alerting it to the action or function that sent the data stream. Passed by reference.
event-flag	An event flag to be set upon completion. Passed by reference.
ast-addr	A user-written procedure called by the application 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 SNA3270$TRANSMIT_SCREEN procedure can return the following status messages.

- SNA3270$_OK
- SNA3270$_INVID
- SNA3270$_NYTXMIT
- SNA3270$_RCVPEND
- SNA3270$_REQREJECT
- SNA3270$_SCRACT

- SNA3270$_XMITSCRFAI

When you write an application for the 3270 Data Stream Programming Interface, the application needs to check for the different status messages that the Interface can return to the application. For example, in some procedures, if the application receives the SNA3270$_NYTXMIT (not your turn to transmit), it cannot issue a SNA3270$TRANSMIT_STREAM procedure. If the application checks only for an SNA3270$_OK status at this point, it will miss the SNA3270$_NYTXMIT and it may try to issue a SNA3270$TRANSMIT_STREAM. If the application tries to issue a SNA3270$TRANSMIT_STREAM at this time, you may receive status messages from the Interface that you do not expect. Therefore, you need to check for the particular status code returned and not just for the SNA3270$_OK status. For further information about the success status messages, refer to Section D.1.

6.10 SNA3270$TRANSMIT_SIGNAL

The SNA3270$TRANSMIT_SIGNAL procedure causes the Interface to request the IBM host to allow the OpenVMS application to transmit again without first receiving data from IBM. This procedure always completes synchronously. It does not wait for the IBM application to respond to the request. See Section 3.3.1 for more information about synchronous completion.

Format:

status.wlc.v=SNA3270$TRANSMIT_SIGNAL (*session-id.rlu.r*,
status-vec.wx.dx,
[*data-bufr.rx.dx*])

Arguments:

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

session-id The session identifier assigned at connect time. Passed by reference.

status-vec A longword vector allocated by the OpenVMS application and filled in by the Interface to provide the user with complete status information. Passed by descriptor.

data-bufr This is an optional parameter. If you specify the parameter, the Interface sends the contents of the buffer. You can send a maximum of four bytes of data. If you do not specify the parameter, the Interface sends the signal request. Passed by descriptor.

The SNA3270$TRANSMIT_SIGNAL procedure can return the following status messages.

- SNA3270$_OK
- SNA3270$_BUFLARGE
- SNA3270$_INVSID
- SNA3270$_NOTINFMOD
- SNA3270$_XMITSIGFAI

When you write an application for the 3270 Data Stream Programming Interface, the application needs to check for the different status messages that the Interface can return to the application. For example, in some procedures, if the application receives the SNA3270$_NYTXMIT (not your turn to transmit), it cannot issue a SNA3270$TRANSMIT_STREAM procedure. If the application checks only for an SNA3270$_OK status at this point, it will miss the SNA3270$_NYTXMIT and it may try to issue a SNA3270$TRANSMIT_

STREAM. If the application tries to issue a SNA3270$TRANSMIT_STREAM at this time, you may receive status messages from the Interface that you do not expect. Therefore, you need to check for the particular status code returned and not just for the SNA3270$_OK status. For further information about the success status messages, refer to Section D.1.

6.11 SNA3270$TRANSMIT_STREAM

The SNA3270$TRANSMIT_STREAM procedure transmits a complete 3270 data stream to the IBM host.

Format:

status.wlc.v=SNA3270$TRANSMIT_STREAM[_W] (*session-id.rlu.r,*
status-vec.wz.dx,
[*data-bufr.rr.dx*],
[*data-length.rlu.r*],
[*last-flag.rlu.r*],
[*event-flag.rlu.r*],
[*ast-addr.zem.r*],
[*ast-par.rlu.r*])

Arguments:

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

session-id The session identifier assigned at connect time. Passed by reference.

status-vec A longword vector allocated by the OpenVMS application and filled in by the Interface to provide the user with complete status information. Passed by descriptor.

data-bufr A data buffer containing a 3270 data stream. The data placed into the buffer must be offset by a value of SNA3270$K_BUF_HDLEN to prevent the Interface header from overwriting the data stream (see Section 2.3.1). Passed by descriptor.

data-length The length of the data to be transmitted including the Interface header. The length must be less than or equal to the size of the buffer. If the length is zero, the interface transmits the entire contents of the buffer. Passed by reference.

last-flag A flag set to indicate whether a transmission is made of single or of multiple calls. If a buffer contains a complete transaction, signal the end of data with the SNA3270$K_END_OF_DATA flag. If the transaction requires more than one buffer, signal that more data is coming with the SNA3270$K_MORE_DATA flag. Be sure to signal the last buffer of a multiple buffer transaction with the SNA3270$K_END_OF_DATA flag. Passed by reference.

event-flag An event flag to be set at completion. Passed by reference.

ast-addr A user-written procedure called by the application 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 SNA3270$TRANSMIT_STREAM procedure can return the following status messages.

- SNA3270$_OK
- SNA3270$_INVSID
- SNA3270$_NYTXMIT
- SNA3270$_RCVPEND
- SNA3270$_REQREJECT
- SNA3270$_XMITSTFAI

When you write an application for the 3270 Data Stream Programming Interface, the application needs to check for the different status messages that the Interface can return to the application. For example, in some procedures, if the application receives the SNA3270$_NYTXMIT (not your turn to transmit), it cannot issue a SNA3270$TRANSMIT_STREAM procedure. If the application checks only for an SNA3270$_OK status at this point, it will miss the SNA3270$_NYTXMIT and it may try to issue a SNA3270$TRANSMIT_STREAM. If the application tries to issue a SNA3270$TRANSMIT_STREAM at this time, you may receive status messages from the Interface that you do not expect. Therefore, you need to check for the particular status code returned and not just for the SNA3270$_OK status. For further information about the success status messages, refer to Section D.1.

6.12 SNA3270$WRITE_FIELD

The SNA3270$WRITE_FIELD procedure writes a field in a 3270 screen image.

Format:

	(*session-id.rlu.r*,
status.wlc.v=SNA3270$WRITE_FIELD	*status-vec.wz.dx*,
	[*data-bufr.rt.dx*])

Arguments:

status	When a procedure finishes execution, it returns a numeric status value in general register R0. Successful completion is indicated by a status code with the low-order bit set. The low-order three bits together represent the severity of the error. Returned as a function value.
session-id	The session identifier assigned at connect time. Passed by reference.
status-vec	A longword vector allocated by the OpenVMS application and filled in by the Interface to provide the user with complete status information. Passed by descriptor.
data-bufr	A data buffer containing the EBCDIC text that is written into the screen image. Passed by descriptor.

The SNA3270$WRITE_FIELD procedure can return the following status messages.

- SNA3270$_OK
- SNA3270$_OK_TRUNC
- SNA3270$_BADOFFSET
- SNA3270$_INVSID
- SNA3270$_NOINPUT
- SNA3270$_NOINFMOD
- SNA3270$_NUMERIC
- SNA3270$_PROTECTED
- SNA3270$_WTFLDFAI

When you write an application for the 3270 Data Stream Programming Interface, the application needs to check for the different status messages that the Interface can return to the application. For example, in some procedures, if the application receives the SNA3270$_NYTXMIT (not your turn to transmit), it cannot issue a SNA3270$TRANSMIT_STREAM procedure. If the application checks only for an SNA3270$_OK status at this point, it will miss the SNA3270$_NYTXMIT and it may try to issue a SNA3270$TRANSMIT_

STREAM. If the application tries to issue a SNA3270$TRANSMIT_STREAM at this time, you may receive status messages from the Interface that you do not expect. Therefore, you need to check for the particular status code returned and not just for the SNA3270$_OK status. For further information about the success status messages, refer to Section D.1.

Part IV

Appendixes

A
Attention Identification Values

The following table provides values for the attention identification (AID) codes that the OpenVMS application sends to the IBM host.

Table A–1 Attention Identification Values

Symbol	Hexadecimal Character (EBCDIC)	Keyboard Equivalent
SNA3270$K_AID_NOAIDD	60	No AID generated (display)
SNA3270$K_AID_NOAIDP	E8	No AID generated(print)
SNA3270$K_AID_ENTER	7D	Press ENTER key
SNA3270$K_AID_PF1	F1	Press PF1 key
SNA3270$K_AID_PF2	F2	Press PF2 key
SNA3270$K_AID_PF3	F3	Press PF3 key
SNA3270$K_AID_PF4	F4	Press PF4 key
SNA3270$K_AID_PF5	F5	Press PF5 key
SNA3270$K_AID_PF6	F6	Press PF6 key
SNA3270$K_AID_PF7	F7	Press PF7 key
SNA3270$K_AID_PF8	F8	Press PF8 key
SNA3270$K_AID_PF9	F9	Press PF9 key
SNA3270$K_AID_PF10	7A	Press PF10 key
SNA3270$K_AID_PF11	7B	Press PF11 key
SNA3270$K_AID_PF12	7C	Press PF12 key
SNA3270$K_AID_PF13	C1	Press PF13 key

(continued on next page)

Table A–1 (Cont.) Attention Identification Values

Symbol	Hexadecimal Character (EBCDIC)	Keyboard Equivalent
SNA3270$K_AID_PF14	C2	Press PF14 key
SNA3270$K_AID_PF15	C3	Press PF15 key
SNA3270$K_AID_PF16	C4	Press PF16 key
SNA3270$K_AID_PF17	C5	Press PF17 key
SNA3270$K_AID_PF18	C6	Press PF18 key
SNA3270$K_AID_PF19	C7	Press PF19 key
SNA3270$K_AID_PF20	C8	Press PF20 key
SNA3270$K_AID_PF21	C9	Press PF21 key
SNA3270$K_AID_PF22	4A	Press PF22 key
SNA3270$K_AID_PF23	4B	Press PF23 key
SNA3270$K_AID_PF24	4C	Press PF24 key
SNA3270$K_AID_SLPA	7E	Selector-light-pen attention
SNA3270$K_AID_PA1	6C	Press PA1 key
SNA3270$K_AID_PA2	6E	Press PA2 key
SNA3270$K_AID_PA3	6B	Press PA3 key
SNA3270$K_AID_CLEAR	6D	Press CLEAR key
SNA3270$K_AID_REQ	F0	Press REQUEST key

B

Summary of Procedure Parameter Notation

This appendix summarizes the notation used to describe parameters in the Digital SNA 3270 Data Stream Programming Interface. For further information about notations and their definitions, see the "OpenVMS' 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 parameter.
2. <Access type> is a single letter denoting the type of access that the procedure will (or can) make to the argument.
3. <Data type> is a letter denoting the primary data type with trailing qualifier letters to identify the data type further. The routine must reference only the size specified to avoid improper access violations.
4. <Pass mech> 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

<pass mech>

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

C

Definitions for the 3270 Data Stream Programming Interface

The following table presents symbols, values, and meanings to use when you write your application. Digital recommends that you use the definition files that accompany the Interface. This will insulate you from changes made in future releases of the product. Definition files, however, are not provided for every language. If the language you plan to use does not have a definition file, use the information in the following table to write your application.

Table C–1 Definitions for the 3270 Data Stream Programming Interface

Symbol	Value	Meaning
SNA3270$K_ACK_ACCEPT	0	Accept +RSP
SNA3270$K_ACK_INTREQ	134348800	Intervention required -RSP
SNA3270$K_ACK_NOCATG	268894208	Category not supported -RSP
SNA3270$K_ACK_NOFUNC	268632064	Function not supported -RSP
SNA3270$K_ACK_NOPROC	135004160	Procedure not supported
SNA3270$K_ACK_PRMERR	268763136	Parameter error
SNA3270$K_ACK_PSILOS	136970240	Presentation space integrity lost
SNA3270$K_ACK_SLUBUS	137166848	SLU busy
SNA3270$K_ACTIVE	1	Active connect—CON_TYP
SNA3270$K_AID_CLEAR	109	CLEAR (_)
SNA3270$K_AID_ENTER	125	ENTER (')
SNA3270$K_AID_MAX_PA	110	High end of PA key codes (for read)
SNA3270$K_AID_MIN_PA	107	Low end of PA key codes (for read)
SNA3270$K_AID_NONE	96	No AID pressed (-)

(continued on next page)

Table C–1 (Cont.) Definitions for the 3270 Data Stream Programming Interface

Symbol	Value	Meaning
SNA3270$K_AID_PA1	108	PA 1 (%)
SNA3270$K_AID_PA2	110	PA 2 (>)
SNA3270$K_AID_PA3	107	PA 3 (,)
SNA3270$K_AID_PF1	241	PF 1 (1)
SNA3270$K_AID_PF2	242	PF 2 (2)
SNA3270$K_AID_PF3	243	PF 3 (3)
SNA3270$K_AID_PF4	244	PF 4 (4)
SNA3270$K_AID_PF5	245	PF 5 (5)
SNA3270$K_AID_PF6	246	PF 6 (6)
SNA3270$K_AID_PF7	247	PF 7 (7)
SNA3270$K_AID_PF8	248	PF 8 (8)
SNA3270$K_AID_PF9	249	PF 9 (9)
SNA3270$K_AID_PF10	122	PF10 (:)
SNA3270$K_AID_PF11	123	PF11 (#)
SNA3270$K_AID_PF12	124	PF12 (@)
SNA3270$K_AID_PF13	193	PF13 (A)
SNA3270$K_AID_PF14	194	PF14 (B)
SNA3270$K_AID_PF15	195	PF15 (C)
SNA3270$K_AID_PF16	196	PF16 (D)
SNA3270$K_AID_PF17	197	PF17 (E)
SNA3270$K_AID_PF18	198	PF18 (F)
SNA3270$K_AID_PF19	199	PF19 (G)
SNA3270$K_AID_PF20	200	PF20 (H)
SNA3270$K_AID_PF21	201	PF21 (I)
SNA3270$K_AID_PF22	74	PF22 (¢)
SNA3270$K_AID_PF23	75	PF23 (.)
SNA3270$K_AID_PF24	76	PF24 (<)
SNA3270$K_AID_TESTREQ	240	TEST REQ (0)

(continued on next page)

Table C–1 (Cont.) Definitions for the 3270 Data Stream Programming Interface

Symbol	Value	Meaning
SNA3270$K_ATTR_HIGH	2	High intensity, detectable
SNA3270$K_ATTR_INVIS	3	Nondisplayed, nondetectable
SNA3270$K_ATTR_LENGTH	2	Data structure size
SNA3270$K_ATTR_NORM	0	Normal intensity, nondetectable
SNA3270$K_ATTR_PEN_DET	1	Normal intensity, detectable
SNA3270$K_BUF_HDLEN	7	Header size
SNA3270$K_CMD_ERALLUNP	111	Erase all unprotected fields
SNA3270$K_CMD_ERWRITE	245	Erase/write all fields
SNA3270$K_CMD_ERWRITEALT	126	Erase/write all fields, use alternate screen size
SNA3270$K_CMD_NULL	0	No command seen
SNA3270$K_CMD_READ	242	Read buffer
SNA3270$K_CMD_READMOD	246	Read modified field
SNA3270$K_CMD_READMODALL	110	Read modified all fields
SNA3270$K_CMD_WRITE	241	Write
SNA3270$K_CMD_WRITESTRF	243	Write structured field
SNA3270$K_EVT_CLEAR	1	Session has received a CLEAR
SNA3270$K_EVT_DATA	2	Data arrived from PLU
SNA3270$K_EVT_MAX	7	Maximum value for event code
SNA3270$K_EVT_MIN	1	Minimum value for event code
SNA3270$K_EVT_PROPROERR	3	SNA protocol error detected
SNA3270$K_EVT_RECONNECTED	4	Reconnected
SNA3270$K_EVT_TERMINATE	5	Session terminated
SNA3270$K_EVT_TURNGONE	6	Turn to send taken by PLU
SNA3270$K_EVT_UNBINDT2	7	Received an UNBIND type 2
SNA3270$K_FDB_SIZE	18	Field descriptor block size
SNA3270$K_FIELD_MODE	1	Field mode session MODE_TYP
SNA3270$K_ORD_CR	13	Carriage return (P)*

(continued on next page)

Table C–1 (Cont.) Definitions for the 3270 Data Stream Programming Interface

Symbol	Value	Meaning
SNA3270$K_ORD_DUP	28	Duplicate (A)*
SNA3270$K_ORD_EM	25	End message (P)*
SNA3270$K_ORD_EUA	18	Erase unprotected to address*
SNA3270$K_ORD_FF	12	Formfeed (P)*
SNA3270$K_ORD_FM	30	Field mark (A)*
SNA3270$K_ORD_GE	8	Graphics escape*
SNA3270$K_ORD_HYPHEN	96	Hyphen character (A)*
SNA3270$K_ORD_IC	19	Insert cursor*
SNA3270$K_ORD_MAX	63	High end of order range*
SNA3270$K_ORD_MF	44	Modify field*
SNA3270$K_ORD_MIN	1	Low end of order code range*
SNA3270$K_ORD_NL	21	New line (P)*
SNA3270$K_ORD_NUL	0	Null (A)*
SNA3270$K_ORD_PT	5	Program tab*
SNA3270$K_ORD_RA	60	Repeat to address*
SNA3270$K_ORD_SA	40	Set attribute*
SNA3270$K_ORD_SBA	17	Set buffer address*
SNA3270$K_ORD_SF	29	Start field*
SNA3270$K_ORD_SFE	41	Start field extended*
SNA3270$K_ORD_SPACE	64	Space character (A)*
SNA3270$K_ORD_SUB	63	Substitution character*
SNA3270$K_PASSIVE	2	Passive connect—CON_TYP
SNA3270$K_SDB_LENGTH	12	Length of context block
SNA3270$K_SEL_READ	0	Read at specified offset
SNA3270$K_SEL_READ_NEXT	2	Read next field
SNA3270$K_SEL_SEARCH	1	Search from specified offset
SNA3270$K_SEL_SEARCH_NEXT	3	Search starting from next field

(continued on next page)

Table C–1 (Cont.) Definitions for the 3270 Data Stream Programming Interface

Symbol	Value	Meaning
SNA3270$K_STREAM_MODE	2	Data stream mode session MODE_TYP
SNA3270$K_WCC_LENGTH	1	Data structure length

* Write orders can be present in the data field of all write-class commands. You can distinguish them from displayable characters because they have values not greater than 3F (hexadecimal). Orders flagged with (P) are for printer support. Those flagged with (A) are special characters for application program use rather than real 3270 orders.

D
Status Codes

The Digital SNA 3270 Data Stream Interface returns the following four types of status codes:

Success codes	indicate that the intended operation succeeded.
Informational codes	provide additional information about success of the intended operation.
Error codes	indicate that the intended operation failed but recovery is possible.
Fatal error codes	indicate that the intended operation failed but recovery is impossible.

D.1 Success Codes

SNA3270$_OK, normal successful completion

Explanation: When the SNA3270$OK message is returned by the SNA3270$TRANSMIT_SCREEN or SNA3270$TRANSMIT_STREAM procedures, it means that the data was successfully transmitted. When this message is returned by the SNA3270$RECEIVE_SCREEN or SNA3270$RECEIVE_STREAM procedures, it means that the PLU has sent data and relinquished the CDI (it is now your turn to send). The PLU is no longer allowed to send data. You will receive an error if you post a receive.

User Action: You must now transmit.

SNA3270$_OK_CONT, successful completion, now in contention

Explanation: For the moment, the PLU has sent all the data it is going to, and you are now in contention state.

User Action: You can now take one of the following actions:

- Send data with the SNA3270$TRANSMIT_SCREEN or SNA3270$TRANSMIT_STREAM procedures.
- Post a SNA3270$RECEIVE_SCREEN or SNA3270$RECEIVE_STREAM procedure and wait for the PLU to send you data.

- Issue a SNA3270$LOCK_SCREEN procedure, which will prevent the PLU from sending you data. You must now transmit.

See Section 2.3.2.2 for a description of possible communication sequences.

SNA3270$_OK_MORE, successful completion, more data in chain

Explanation: The receive buffer was too small to hold the whole message. A subsequent receive will pick up the rest of the message and complete with one of the other success codes.

User Action: Issue another receive request.

SNA3270$_OK_NYT, successful completion, not your turn to transmit

Explanation: The application has successfully received data and the PLU has more data to send.

User Action: You must issue another receive. If you attempt to transmit, the transmit will fail and will return a SNA3270$_NYTXMIT message (not your turn to transmit).

SNA3270$_OK_TRUNC, data truncated, destination string too small

Explanation: The read or write field procedure completed normally but the destination buffer was not large enough to receive all the data specified.

D.2 Informational Codes

SNA3270$_CLEARREC, CLEAR received, data traffic now reset

Explanation: The IBM system has sent a clear command.

User Action: Enter the data traffic reset state. See the network manager for more information.

SNA3270$_DATAREC, data received, issue a SNA3270$RECEIVE_STREAM

Explanation: Data has been sent by the IBM system.

User Action: Issue a RECEIVE to receive data.

SNA3270$_RECINPR, UNBIND received, reconnection in progress

Explanation: The Interface is attempting to reestablish the session on your behalf.

User Action: Your notify routine will be called when the reconnection completes. No data can be transmitted or received until the session is reconnected.

SNA3270$_RECONNECTED, session has been reconnected

Explanation: The Interface has successfully reconnected the session.

User Action: You can resume transmitting and receiving data.

D.3 Error Codes

SNA3270$_ACKFAI, failed to acknowledge data

Explanation: The acknowledge request has failed.

User Action: See the status vector for more information.

SNA3270$_ATTRSHO, ATTR_VEC is too short, it must be at least '*nn*' bytes long

Explanation: The attributes vector is too short to accommodate the building of a screen image.

User Action: Allocate a larger amount of memory for the attributes vector and reissue the command. The maximum screen size field in the SDB contains the required size in words.

SNA3270$_BADOFFSET, buffer offset in FDB is not the start of field

Explanation: The buffer offset specified in the field descriptor block is not the start of a field.

User Action: Change the offset to be the start of a field, then reissue the command. Alternatively, you may want to change the search mode to READ_NEXT or SEARCH.

SNA3270$_BADVEC, bad vector descriptor

Explanation: A bad vector descriptor has been supplied as a parameter.

User Action: The vector was improperly specified or has been corrupted, or the BIND parameters require a size larger than that specified.

SNA3270$_BUFLARGE, SIGNAL message cannot be larger than four bytes

Explanation: The buffer specified in the message is too large.

User Action: Decrease the buffer to 4 bytes or less.

SNA3270$_BUFSMALL, buffer must be at least SNABUF$K_HDLEN + 1 bytes long

Explanation: The buffer specified in the receive request is too small.

User Action: Increase the buffer to at least the request unit (RU) size. The RU size can be obtained from your IBM system programmer.

SNA3270$_CHARSHO, CHAR_VEC is too short, it must be at least '*nn*' bytes long

Explanation: The character vector is too short to accommodate the building of a screen image.

User Action: Allocate a larger amount of memory for the character vector and reissue the command.

SNA3270$_CLEAR, request aborted due to CLEAR command

Explanation: A clear request was received from the PLU.

User Action: The application must disconnect and reestablish the session. If the condition persists, the PLU is detecting protocol or data stream errors. Use the SNA Trace facility to isolate the problem.

SNA3270$_CONFAI, connect failed

Explanation: A connect request failed.

User Action: See the status vector for more information.

SNA3270$_DISFAI, disconnect failed

Explanation: A disconnect request failed.

User Action: See the status vector for more information.

SNA3270$_EXIT, gateway server task terminated

Explanation: The cooperating software in the DECnet SNA Gateway has failed.

User Action: Look for log messages on the operator's console of the Gateway's loading host. (See your system or network manager for more information.)

SNA3270$_FDBLENERR, the length of the field descriptor block is incorrect

Explanation: The length of the field descriptor block is incorrect.

User Action: Adjust the application logic and rerun.

SNA3270$_FVECSHO, FIELD_VEC is too short, it must be at least '*nn*' bytes long

Explanation: The field vector is too short.

User Action: Allocate a larger amount of memory for the field vector, then reissue the command.

SNA3270$_GATCOMERR, error communicating with Gateway node

Explanation: A fatal communication error has occurred and your session is lost.

User Action: See the secondary error code that accompanies this message for further information. If you require more information, see your network manager.

SNA3270$_ILLASTSTA, illegal AST state

Explanation: You have issued a synchronous procedure call from within an AST procedure.

User Action: You must restructure your application.

SNA3270$_ILLCONTYP, illegal connection type

Explanation: The CONN_TYP parameter value in the SNA3270$REQUEST_CONNECT procedure call was not equal to either SNA3270$K_ACTIVE or SNA3270$K_PASSIVE.

User Action: Issue SNA3270$REQUEST_CONNTECT with CONN_TYP equal to either SNA3270$K_ACTIVE or SNA3270$K_PASSIVE.

SNA3270$_ILLMODTYP, illegal mode type

Explanation: The MODE_TYP parameter value in the SNA3270$REQUEST_CONNECT procedure call was not equal to either SNA3270$K_FIELD_MODE or SNA3270$K_STREAM_MODE.

User Action: Issue SNA3270$REQUEST_CONNTECT with MODE_TYP equal to either SNA3270$K_FIELD_MODE or SNA3270$K_STREAM_MODE.

SNA3270$_INVSID, invalid session ID

Explanation: No session corresponding to passed SESSION_ID.

User Action: Either the session is already inactive or an incorrect SESSION_ID was supplied.

SNA3270$_LOCFAI, lock screen failed

Explanation: The lock screen request failed.

User Action: See whether you have received an end bracket indicator (EBI). If you have not received an EBI, you may need to wait for one. See the status vector for more information.

SNA3270$_NEGRSP, negative response received, sense code %X'*nn*'

Explanation: A transmission has been rejected by the IBM system.

User Action: Determine why data is rejected on the basis of the sense code, adjust the application logic, and rerun the program.

SNA3270$_NETSHUT, network node is not accepting connects

Explanation: The Gateway DECnet executive state has changed to SHUT or OFF.

User Action: See your system or network manager to determine why the Gateway is not available.

SNA3270$_NOFIELD, no field was found

Explanation: The attempt to read or write a field in the character vector failed because the field specified by way of the attribute and offset fields in the FDB either does not exist or was not found before the end of the character vector was detected.

User Action: Reset the starting position to 0 and search for the field.

SNA3270$_NOINPUT, no input data for WRITE_FIELD verb

Explanation: No data has been provided for a write field verb.

User Action: Adjust the application logic and rerun.

SNA3270$_NORSPPEND, no response is pending

Explanation: The SNA3270$ACKNOWLEDGE procedure was called when the PLU was not expecting a response. None was sent.

User Action: Check the application logic.

SNA3270$_NOTINFMOD, not in field mode

Explanation: This procedure is valid in field mode only.

User Action: Determine correct mode (SNA3270$K_FIELD_MODE for field mode) and set the CONN_MODE parameter accordingly in the connect.

SNA3270$_NUMERIC, data must be numeric

Explanation: An attempt was made to write nonnumeric data in a numeric field.

User Action: Check and adjust application logic, then rerun.

SNA3270$_NYTRCV, not your turn to receive data

Explanation: It is not your turn to receive data.

User Action: Your application is in the transmit state; check and adjust the application logic, then rerun.

SNA3270$_NYTXMIT, not your turn to transmit data

Explanation: It is not your turn to transmit data.

User Action: Your application is in the receive state; you must keep issuing receives until it completes with OK or OK_CONT. Check and adjust the application logic, then rerun.

SNA3270$_PROTECTED, field is protected, no modification allowed

Explanation: An attempt was made to write data to a protected field.

User Action: Check and adjust the application logic, then rerun program.

SNA3270$_PROTERR, SNA protocol error

Explanation: A protocol error has occurred in the SNA layer.

User Action: See the secondary error code that accompanies this message for further information. If you require more information, see your network manager.

SNA3270$_RCVPEND, call not allowed while a receive is outstanding

Explanation: You have issued a transmit request while a receive is pending.

User Action: Wait for the receive to complete.

SNA3270$_RDFLDFAI, failed to read field

Explanation: The read field request has failed.

User Action: See the status vector for more information.

SNA3270$_RECSCRFAI, failed to receive a screen of data

Explanation: The receive screen request has failed.

User Action: See the status vector for more information.

SNA3270$_RECVSTFAI, failed to receive 3270 data stream

Explanation: The receive stream request has failed.

User Action: See the status vector for more information.

SNA3270$_REQREJECT, invalid data received, rejected with sense code %X'*nn*'

Explanation: Invalid 3270 data was received from the IBM system.

User Action: See your network manager for more information.

SNA3270$_SCRACT, previous TRANSMIT_SCREEN/RECEIVE_SCREEN has not completed

Explanation: A transmit or receive screen has been attempted before the previous one has been completed.

User Action: Wait for the previous transmit or receive request to complete, then retry.

SNA3270$_SCRLENERR, SDB length is incorrect

Explanation: The screen descriptor block length is incorrect.

User Action: Check and adjust the application logic, then rerun.

SNA3270$_SESTERM, session terminated

Explanation: The session has been terminated.

User Action: See the notify vector for more information.

SNA3270$_TURNGONE, not your turn to send anymore

Explanation: A BID has been sent by the IBM system and accepted by the Interface.

User Action: Issue a receive request to receive the incoming data. Alternatively, you can use the lock screen command to prevent this from happening by redesigning the application logic.

SNA3270$_UNABD0, unacceptable BIND image, byte *N* field name *xxx*

Explanation: The data described was not acceptable.

User Action: Your IBM system programmer must redefine the parameters in error as specified in the *DECnet SNA Gateway-CT Guide to IBM Parameters*, the *DECnet SNA Gateway-ST Guide to IBM Parameters*, or the *Digital SNA Domain Gateway Guide to IBM Resource Definition* manual.

SNA3270$_UNABD1, unacceptable BIND image, byte *N*, bit *B*, field name *xxx*

Explanation: The data described was not acceptable.

User Action: Your IBM system programmer must redefine the parameters in error as specified in the *DECnet SNA Gateway-CT Guide to IBM Parameters*, the *DECnet SNA Gateway-ST Guide to IBM Parameters*, or the *Digital SNA Domain Gateway Guide to IBM Resource Definition* manual.

SNA3270$_UNABD2, unacceptable BIND image, byte *N*, bits *B1-B2*, field name *xxx*

Explanation: The data described was not acceptable.

User Action: Your IBM system programmer must redefine the parameters in error as specified in the *DECnet SNA Gateway-CT Guide to IBM Parameters*, the *DECnet SNA Gateway-ST Guide to IBM Parameters*, or the *Digital SNA Domain Gateway Guide to IBM Resource Definition* manual.

SNA3270$_UNARANGE, needed a value in range %X'*nn*' to %X'*nn*', received %X'*nn*'

Explanation: A value was received in a BIND image that was out of range.

User Action: The particular BIND field in error is reported in a preceding message.

SNA3270$_UNAVALUE, expected %X'*nnn*'!+, received %X'*nnn*'

Explanation: An unacceptable value was received in a BIND image.

User Action: The particular BIND field in error is reported in a preceding message.

SNA3270$_WTFLDFAI, failed to write field

Explanation: The write field request has failed.

User Action: See the status vector for more information.

SNA3270$_XMITSCRFAI, failed to transmit a screen of data

User Action: The transmit screen request has failed.

Explanation: See the status vector for more information.

SNA3270$_XMITSIGFAI, failed to transmit signal

Explanation: The transmit signal request has failed.

User Action: See the status vector for more information.

SNA3270$_XMITSTFAI, failed to transmit 3270 data stream

Explanation: The transmit stream request failed.

User Action: See the status vector for more information.

D.4 Fatal Error Codes

SNALU3270$_BUGCHK, internal error detected in *routine-name* at PC *nnnnnn*

Explanation: An internal error has been detected.

User Action: Write down all the messages that appear on your screen at this time and report the problem to your system manager.

The following secondary error messages appear along with the top-level SNALU3270$_BUGCHK error. The appearance of any of the following error messages implies that the 3270 Data Stream Programming Interface is operating abnormally. To take corrective action, copy all the messages associated with the fatal error code. Take your list of error messages to your system manager, who can decide what corrective action to take. Consult the *DECnet SNA Gateway Problem Determination Guide*, the *DECnet SNA Gateway-CT Problem Solving (OpenVMS & ULTRIX)*, or the *DECnet SNA Gateway-ST Problem Solving (OpenVMS)* manual, if you are the system manager. If you cannot solve your problem, submit a Software Performance Report (SPR) if you have the service.

SNA3270$_BLKTYP, ’*xx*’ called with invalid block type (value’*nn*’)

SNA3270$_DFCERR, unexpected result from Data Flow Control

SNA3270$_ERRALLBUF, error allocating dynamic buffer

SNA3270$_FMTFMD, formatted FMD data received

SNA3270$_FREEVM, call to LIB$FREE_VM failed

SNA3270$_INVASY, invalid asynchronous event occurred, code ’*nn*’

SNA3270$_INVDFC, invalid Data Flow Control (DFC) RU received, RU code %X’*nn*’

SNA3270$_INSRES, insufficient resources to perform requested operation

SNA3270$_INVSC, invalid session control (SC) RU received, RU code %X'*nn*'

SNA3270$_NCSCREC, network control/session control (NC/SC) RU received on normal flow

SNA3270$_STRFRE, failed to free memory used by a D-type descriptor

SNA3270$_UMBXMSG, unrecognized mailbox message received, code '*nn*'W

E

Low-Level Status Codes

This appendix provides information about error codes and messages that you may receive from lower layers of the Interface.

- General error codes
- General subfailure codes
- Status codes for abort reasons returned from the Gateway
- Fatal error codes

You can find the text for these messages in SYS$MESSAGE:SNA3270MG.EXE.

E.1 General Error Codes

SNA$_MUTORCVCHK, MU generated a receive check, sense code *IBM sense code*

Explanation: The message unit returned a receive check sense code.

User Action: Consult your IBM manual for the sense code.

SNA$_MUTOSENDCHK, MU generated a send check, sense code *IBM sense code*

Explanation: The message unit returned a send check sense code.

User Action: Consult your IBM manual for the sense code.

SNA$_PLUPROVIO, PLU violated SNA protocol rules, sense code *IBM sense code*

Explanation: The primary logical unit violated SNA protocol rules.

User Action: Consult your IBM manual for the sense code.

SNA$_UNABLELUCB, unable to obtain lucb

Explanation: Insufficient virtual memory.

User Action: Increase virtual memory.

SNA$_UNABLEMUCB, unable to obtain mucb

Explanation: Insufficient virtual memory.

User Action: Increase virtual memory.

SNA$_UNABLESCB, unable to obtain scb

Explanation: Insufficient virtual memory.

User Action: Increase virtual memory.

E.2 General Subfailure Codes

SNA$_FAIALLBUF, failed to allocate memory for a buffer

Explanation: The Interface failed to allocate dynamic memory for an internal buffer. The most likely reason is that no free memory is available.

User Action: If you are using class D descriptors, make sure you return used buffers to free memory with LIB$SFREE1_DD or STR$FREE1_DX.

SNA$_FAIALLCTX, failed to allocate memory for a context block

Explanation: The Interface failed to allocate memory for an internal context block. The most likely reason is that no free memory is available.

User Action: If you are using class D descriptors, make sure you return used buffers to free memory with LIB$SFREE1_DD or STR$FREE1_DX.

SNA$_FAIASSCHA, failed to assign a DECnet channel

Explanation: The error indicates an abnormal DECnet condition.

User Action: Examine the subsequent DECnet error messages and report the problem to your system manager.

SNA$_FAIBLDNCB, failed to build DECnet network connect block

Explanation: The Interface failed to build a DECnet network connect block in order to communicate with the Gateway.

User Action: Examine the subsequent error messages for more information.

SNA$_FAICONMBX, failed to convert mailbox name

Explanation: The Interface could not create a mailbox for establishing a logical link.

User Action: Examine subsequent error messages to find the reason. The most likely additional message is SYSTEM-F-NOPRIV, which indicates no privilege for attempted operation. This means that you lack TMPMBX privilege.

SNA$_FAICOPBIN, failed to copy BIND request image into caller's buffer

Explanation: The Interface could not copy the entire BIND request image into the BIND request buffer provided by the application.

User Action: Make sure that you specify a BIND buffer that is large enough to receive the largest BIND that the IBM application will send you.

SNA$_FAIESTLIN, failed to establish a link to the Gateway

Explanation: The Interface cannot connect to the Gateway.

User Action: Examine the subsequent error messages and take appropriate action.

SNA$_FUNCABORT, access routine function aborted

Explanation: The Interface procedure did not complete successfully and the session has been or is being terminated.

User Action: Ignore the error. You have or will get notification of an asynchronous event that will tell you why the session has terminated.

SNA$_FUNNOTVAL, function not valid with port in current state

Explanation: The Interface is invalid with the port in the current state. You issued Interface calls in the wrong order–for example, an SNA$TRANSMIT before an SNA$ACCEPT.

User Action: Correct the code in your application.

SNA$_GATCOMERR, error communicating with Gateway node

Explanation: There was a communication error with the Gateway node.

User Action: Examine the subsequent error messages and take appropriate action.

SNA$_ILLASTSTA, ASTs are disabled or an AST routine is currently in progress

Explanation: A call was made to an Interface procedure 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 Interface procedures are not called from AST routines or with ASTs disabled.

SNA$_INSRESOUR, insufficient resources to establish session

Explanation: The Interface could not allocate enough system resources to establish the session.

User Action: Examine the subsequent messages for more information.

SNA$_INVGWYNOD, parameter GWY-NODE is invalid

Explanation: You entered an invalid value in the GWY-NODE parameter.

User Action: Examine the call that returned the error and take appropriate action.

SNA$_INVRECLOG, SNA$DEF_NUMREC is incorrectly defined

Explanation: This internal logical name is set up improperly.

User Action: SNA$DEF_NUMREC is a logical name that determines the number of receives the Interface keeps outstanding on the DECnet logical link. If you do not wish to use the default value, use the DEFINE command (for example, DEFINE SNA$DEF_NUMREC 5).

SNA$_MAXSESACT, maximum number of sessions already active

Explanation: You have already established 120 sessions, the maximum number allowed.

User Action: Make sure you have called the disconnect procedure for each session that has terminated.

SNA$_NO_GWYNOD,SNA$DEF_GATEWAY is undefined and GWY-NODE was not specified

Explanation: No Gateway node was specified, and the logical name SNA$DEF_GATEWAY was not defined.

User Action: Either supply an explicit Gateway node specification or define SNA$DEF_GATEWAY using the OpenVMS DEFINE command.

E.3 Status Codes for Abort Reasons Returned From Gateway

After the Interface returns any of the following codes, the session is no longer active. The application should issue a disconnect request before attempting to establish another session.

SNA$_ACCINTERR, Gateway detected an error in the Gateway access routines

Explanation: This indicates 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.

SNA$_ABNSESTER, session terminated abnormally

Explanation: Either the link between the Gateway and IBM was lost, or IBM deactivated the physical unit (PU) or the line leading to the Gateway.

User Action: Determine why the link was lost. Retry when the connection to IBM returns.

SNA$_ACCROUFAI, error from Gateway access routine, gateway unknown or unreachable

Explanation: SNA Gateway is unknown or unreachable; Transport list (defined by SNA_TRANSPORT_ORDER logical) is defined incorrectly or Gateway/Host Name 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.

SNA$_APPNOTSPE, IBM application name was not specified

Explanation: In the connect request, you did not specify the IBM application name, and the access name that you used did not supply one either.

User Action: The IBM application must be either explicitly supplied in the parameter list or implicitly supplied through the access name.

SNA$_BINSPEUNA, the BIND image specified unacceptable values

Explanation: The Gateway rejected the BIND request image.

User Action: Run a trace to find out why the Gateway rejected the BIND request. The IBM application could be specifying too large an outbound RU or an illegal FM or TS profile, or it could have sent a pacing value that was out of bounds (Refer to the *Digital SNA Domain Gateway Installation*, the *DECnet SNA Gateway-CT Installation*, the *DECnet SNA Gateway-CT Installation*, or the *Digital Peer Server Installation and Configuration* manual).

SNA$_CONREQREJ, connect request rejected by IBM host, sense code %X'*IBM sense code*'

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.

SNA$_GATINTERR, internal error in Gateway node, code %O'*xx*',subcode %O'*xx*'

Explanation: A fatal error has occurred.

User Action: Report the error to your system manager. Also ensure that the log from the Gateway console is saved, which will have messages of the form:

```
GAS -- Fatal Session Error FSE$xxx
```

SNA$_INCVERNUM, Gateway access routines are incompatible with the Gateway

Explanation: The software on the 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 Gateway and the local system.

SNA$_INSGATRES, insufficient Gateway resources for session establishment

Explanation: The Gateway has insufficient resources to establish a session. The active sessions currently in the Gateway are using the total resources available.

User Action: Wait until some of the sessions have finished, then retry.

SNA$_LOGUNIDEA, SSCP has deactivated the session

Explanation: The IBM SSCP has deactivated the session by sending a DACTLU command. Some applications deactivate sessions by deactivating the logical unit rather than by sending an UNBIND command.

SNA$_NOSUCACC, access name not recognized by Gateway node

Explanation: You specified a nonexistent access name.

User Action: Check with your system manager to determine which access name you need.

SNA$_NOSUCPU, PU name not recognized by Gateway node

Explanation: Either you or the access name you used specified a nonexistent physical unit.

User Action: Check with your system manager to determine which PU name or access name you need.

SNA$_NOSUCSES, session address not recognized by Gateway node

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.

SNA$_PROUNBREC, IBM application detected a protocol error, sense code %X'*IBM sense code*'

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.

SNA$_PUNOTAVA, PU has not been activated

Explanation: The physical unit on the 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 Gateway and the IBM host.

SNA$_PUNOTSPE, PU name was not specified

Explanation: In the connect request you did not specify a physical unit name, and the access name that you used did not supply one either.

User Action: The PU name must be either explicitly supplied in the parameter list or implicitly supplied through the access name.

SNA$_SESINUSE, session address is already in use

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.

SNA$_SESNOTAVA, session address has not been activated

Explanation: The SLU has not been activated from the IBM side.

User Action: Ask the IBM VTAM operator to check the logical unit from the IBM host and activate it if necessary.

SNA$_UNBINDREC, UNBIND request received from IBM application

Explanation: The IBM application has terminated the session by sending a normal UNBIND RU.

SNA$_UNUUNBREC, UNBIND of type %X'type' received from IBM application

Explanation: The IBM application sent the specified type of UNBIND request.

User Action: Determine the meaning of this from the IBM documentation on the UNBIND request and take the appropriate action.

E.4 Fatal Error Codes

SNA$_FATINTERR, internal error in Gateway access routines

Explanation: A fatal error has occurred.

User Action: Write down all the messages that appear on your screen at this time and report the problem to your system manager.

The appearance of any of the following error messages implies that the 3270 Data Stream Programming Interface is operating abnormally. To take corrective action, copy all the messages associated with the fatal error code. Take your list of error messages to your system manager, who can decide what corrective action to take. Consult the *Digital SNA Gateway Problem Determination Guide* if you are the system manager. If you cannot solve your problem, submit a Software Performance Report (SPR) if you have the service.

SNA$_ABOCTXPRE, abort context block present at port deletion time

SNA$_ABOWAIACC, abort attempt while still waiting for IO$_ACCESS

SNA$_ASTBLKZER, ASTBLK to SNA$$IOEVENT is 0

SNA$_BINDATREC, Bind data received in wrong state

SNA$_CANCELFAI, $CANCEL failed

SNA$_CTXBLKINU, no active ports, but context blocks still in use

SNA$_DCLASTFAI, $DCLAST failed

SNA$_DFCFATAL, SNA$RECEIVE failed

SNA$_FAICOPMBX, failed to copy mailbox message to context block

SNA$_FAICOPRH, failed to copy RH parameters to user buffer

SNA$_FAICREMBX, failed to create a mailbox

SNA$_FAIDEAMBX, failed to deassign mailbox channel

SNA$_FAIFREBUF, failed to free data buffer

SNA$_FAIFREEF, LIB$FREE_EF failed

SNA$_FAIFRENCB, failed to free NCB buffer

SNA$_FAIGETCHA, failed to get mailbox characteristics

SNA$_FAIGETMBX, failed to get context block for mailbox message

SNA$_FAITRIBLA, failed to trim blanks off end of node name

SNA$_FLUBUFREC, Flush Buf message received while not flushing

SNA$_GATTRAFAI, Gateway logical name translation failed

SNA$_GETDVIFAI, failed to get NET device characteristics

SNA$_ILLMBXMSG, illegal or unexpected mailbox message type

SNA$_INCOPYERR, internal copy error

SNA$_INMEMCT, insufficient memory for correlation table

SNA$_INSENDERR, internal send error

SNA$_INVARGLEN, invalid argument block length in SNA$$DOWAIT

SNA$_INVRECCHK, invalid port state for receive check

SNA$_JUNKSTATE, FSM state reached contained unrecognized data

SNA$_LIBFREFAI, LIB$FREE_VM failed

SNA$_LIBGETFAI, LIB$GET_VM failed

SNA$_MBXIOSERR, mailbox read failed with an IOSB error

SNA$_MBXREAFAI, mailbox read returned an error

SNA$_NOINTMUCB, no internal mu could be obtained

SNA$_NOMEMRSP, no memory to send negative response

SNA$_NOTNORDAT, nonnormal data message received from Gateway

SNA$_NOTOCCURST, a cannot occur state was reached in FSM

SNA$_OBJTRAFAI, failed to translate object name logical

SNA$_PORREFNON, port data base reference count is not zero

SNA$_PORREFOUT, port data base reference count is out of range

SNA$_PORUNKSTA, port is in an unknown state

SNA$_PROBFREEMEM, problem while freeing memory

SNA$_PROBINREVT, problem with internal SNA$READEVENT call

SNA$_PROERRBIN, protocol error in BIND data message from Gateway

SNA$_RECBUFINU, no active ports, but receive buffers still in use

SNA$_RECFREFAI, failed to free receive buffer

SNA$_RECPENMSG, RECONPEND message received, state not RUNNING

SNA$_SNAASSFAI, failed to assign an I/O channel to _SNA0

Explanation: You did not specify a Gateway DECnet node name or TCP/IP host name when using the SNA Gateway.

User Action: Specify a Gateway DECnet node name or TCP/IP host name for the Gateway you want to use.

SNA$_STANOTRUN, normal message received, state not RUNNING

SNA$_UNABLEEVT, unable to obtain DFC event block

SNA$_UNDEFSENDCHK, undefined send check encountered

SNA$_UNDQAST, unable to remqueue ast to process

SNA$_UNFREECT, unable to free correlation table

SNA$_UNFREEEVT, unable to free DFC event block

SNA$_UNFREELUCB, unable to free lucb

SNA$_UNFREEMUCB, unable to free mucb

SNA$_UNFREESCB, unable to free scb

SNA$_UNINUMUNK, unit number in mailbox message is unknown

SNA$_UNKDATMSG, unknown data message type received

SNA$_UNKMSGREC, unknown message code received from Gateway

SNA$_UNKUNBREC, unknown UNBIND type received from Gateway

SNA$_UNQAST, unable to insqueue ast to process

SNA$_UNSUSEREC, unsatisfied user receives at port deletion time

F

Correlation of Procedures and Status Messages for the 3270 Data Stream

The following table illustrates the correlation between the 3270 Data Stream Programming Interface procedures and the status messages they can return.

Table F–1 Procedures and Status Messages Correlation for the 3270 Data Stream

Status	Procedure Keys										
Message	1	2	3	4	5	6	7	8	9	10	11
SNA3270$_OK	√	√	√	√	√	√	√	√	√	√	√
SNA3270$_OK_CONT				√	√						
SNA3270$_OK_MORE					√						
SNA3270$_OK_NYT				√	√						
SNA3270$_OK_TRUNC			√								√
SNA3270$_ACKFAI	√										
SNA3270$_ATTRSHO						√					
SNA3270$_BADOFFSET			√								
SNA3270$_BADVEC						√					
SNA3270$_BUFLARGE									√		
SNA3270$_BUFSMAL					√						
SNA3270$_CHARSHO						√					
SNA3270$_CONFAI						√					
SNA3270$_DISFAI							√				
SNA3270$_FDBLENERR						√					
SNA3270$_FVECSHO						√					
SNA3270$_ILLCONTYP						√					
SNA3270$_ILLMODTYP						√					
SNA3270$_INVSID	√	√	√	√	√	√	√	√	√	√	√
SNA3270$_LOCFAI		√									
SNA3270$_NOFIELD			√								
SNA3270$_NOINPUT											√
SNA3270$_NORSPPEND	√										
SNA3270$_NOTINFMOD			√	√					√		√
SNA3270$_NUMERIC											√

Procedure Keys

1. Acknowledge
2. Lock Screen
3. Read Field
4. Receive Screen
5. Receive Stream
6. Request Connect
7. Request Disconnect
8. Transmit Screen
9. Transmit Signal
10. Transmit Stream
11. Write Field

(continued on next page)

Table F–1 (Cont.) Procedures and Status Messages Correlation for the 3270 Data Stream

Status	Procedure Keys										
Message	1	2	3	4	5	6	7	8	9	10	11
SNA3270$_NYTRCV				√	√						
SNA3270$_NYTXMIT								√		√	
SNA3270$_PROTECTED											√
SNA3270$_RCVPEND								√		√	
SNA3270$_RDFLDFAI			√								
SNA3270$_RECSCRFAI				√							
SNA3270$_RECVSTFAI					√						
SNA3270$_RECREJECT								√		√	
SNA3270$_SCRACT				√				√			
SNA3270$_SCRLENERR						√					
SNA3270$_UNABD0						√					
SNA3270$_UNABD1						√					
SNA3270$_UNABD2						√					
SNA3270$_UNARANGE						√					
SNA3270$_UNAVALUE						√					
SNA3270$_WTFLDFAI											√
SNA3270$_XMITSCRFAI								√			
SNA3270$_XMITSIGFAI									√		
SNA3270$_XMITSTFAI										√	

Procedure Keys

1. Acknowledge
2. Lock Screen
3. Read Field
4. Receive Screen
5. Receive Stream
6. Request Connect
7. Request Disconnect
8. Transmit Screen
9. Transmit Signal
10. Transmit Stream
11. Write Field

Index