IBM

Series/1

SL23-0106-0

EDX Communications Facility Programmer's Guide

Version 2

Introduction	Design and Installation Guide	Operator's Guide
Master Index and Glossary	Programmer's Guide	Messages and Codes
WSC High-Level Language Subroutines Programmer's Guide	Debugging Guide	Operator's Reference Summary

IBM

Series/1

SL23-0106-0

EDX Communications Facility Programmer's Guide

Version 2

Introduction	Design and Installation Guide	Operator's Guide
Master Index and Glossary	Programmer's Guide	Messages and Codes
WSC High-Level Language Subroutines Programmer's Guide	Debugging Guide	Operator's Reference Summary

First Edition (September 1984)

This edition applies to Version 2.0 of the Licensed Program IBM Series/1 Event Driven Executive Communications Facility, Program Number 5719-CF2, and to all subsequent versions and modifications of this program unless otherwise indicated in new editions or technical newsletters.

About This Book

This book is intended for programmers who are going to code application programs, device-support programs, transaction-processing programs, and command-processing programs to run under the IBM Series/1 Event Driven Executive Communications Facility. It assumes that you already understand the Communications Facility's functions, know how a Communications Facility system is organized, and know the purpose of the program you plan to code. If you need introductory information about the Communications Facility, refer to *IBM Series/1 EDX Communications Facility Introduction,* GL23-0071. For detailed information about the structure of a Communications Facility system, see *IBM Series/1 EDX Communications Facility Design and Installation Guide,* SL23-0073.

Other Communications Facility manuals you may want to refer to are *IBM Series/1 EDX Communications Facility Operator's Guide,* SL23-0075, which explains how to use the operator commands and utilities, the *IBM Series/1 EDX Communications Facility Debugging Guide,* LL23-0076, which gives the formats of control blocks and describes the EDL language extensions that are intended for internal system use, and the *IBM Series/1 EDX Communications Facility Messages and Codes,* SL23-0120.

This book assumes that you know the Event Driven Executive Language (EDL) presented in the *IBM Series/1 Event Driven Executive Language Reference,* SC34-0442.

This book also assumes that you know how to use the facilities of the EDX system, as explained in these books:

IBM Series/1 Event Driven Executive Operator Commands and Utilities Reference, SC34-0444

IBM Series/1 Event Driven Executive Messages and Codes, SC34-0445

IBM Event Driven Executive Language Programming Guide, SC34-0438.

If you're creating 3270 data streams, you'll need to refer to:

IBM 3270 Information Display System: Data Stream Programmer's Reference, GA23-0059, or

IBM 3270 Information Display System: 3271 Control Unit, 3272 Control Unit, 3275 Display Station Description and Programmer's Guide, GA23-0060.

If you're writing X.25 application programs, you may need to refer to *IBM Series/1 X.25/HDLC Communications Support General Information Manual,* GC09-1026. If the programs send and receive X.25 control messages, you'll also need to refer to CCITT Recommendation X.25 *Data Communication Networks,* CCITT Yellow Book, volume VIII.2, Geneva, Switzerland ITU, 1981.

You call on the Communications Facility through language extensions and utility programs. This book is intended to give you the information you need to use the utilities and to code programs that use the language extensions.

To that end, it has these chapters:

- "Using the Communications Facility" on page 1 explains the functions you can perform, in a user program, through the Communications Facility's extensions to the Event Driven Executive Language.

- "Writing Communications Facility Programs" on page 27 contains programming information that you'll need to write a Communications Facility program.

- "Creating and Managing 3270 Panels" on page 45 explains how to use the $.PANEL utility program (a part of the Communications Facility) to create the panels that are displayed as part of interactive programs used by operators working at 3270 terminals or Series/1 terminals being managed as if they were 3270 terminals. It includes a sample program.

- "Creating a Transaction" on page 67 explains the coding conventions you must follow in a program that is called in response to a transaction, and the procedures necessary to get such a program installed and running.

- "Creating a Program to Communicate with EDX Terminals" on page 85 explains how to use the work session controller (a part of the Communications Facility) to write an interactive program to be used by operators working at EDX terminals. It also explains how to store images in the work session controller's image library, and how to use the program $.WSMENU to start communication between your interactive program and its users. It includes a sample transaction-processing application.

- "Writing an X.25 Application Program" on page 107 contains the information you'll need to write an application that uses the X.25 protocols.

- "Creating a Command" on page 155 explains how to create a new operator command and make it available to users. It includes a sample command-processing program.

- "Creating an I/O Control Program" on page 165 explains how to structure and code a program to add device support for devices not directly supported by the Communications Facility, and how to get such a program installed and running. It includes a coding example.

- "Coding Communications Facility Instructions" on page 175 presents reference material about each instruction—its format, operands, and return codes.

- "Coding Work Session Controller Transactions" on page 305 presents reference material about the trnasactions you use to code work session controller applications.

- Appendix A, "$.CFMENU Sample Program" on page 375 discusses the $.CFMENU program—a sample application program, distributed as part of the Communications Facility, that demonstrates how to communicate with users at 3270-type terminals. The chapter includes a listing of the sample program.

- The "Glossary" presents the definitions of technical Communications Facility terms and acronyms. For EDX definitions, see the appropriate EDX book.

Summary of Amendments

Only changes affecting the content of this book are listed in the Summary of Amendments.

Version 2.0

The following changes were made between the Communications Facility Version 1.2 and Version 2.0:

Recommendation X.25 Support: The Communications Facility X.25 IOCP supports packet-level communication over network interfaces as defined in CCITT Recommendation X.25. The new IOCP, $.IO0AB8, is described in "Writing Communications Facility Programs" on page 27.

A new chapter is added to this book—"Writing an X.25 Application Program" on page 107.

4980 Terminal Support: The Communications Facility now supports the 4980 terminal.

$.PNLUT1 Utility Program: $.PNLUT1, a utility program that prints descriptions of panels created by $.PANEL, is added to "Creating and Managing 3270 Panels" on page 45.

PUT TCB Instruction: The new programming instruction "PUT TCB—Create a Task Control Block" on page 257 has been added.

SEND SM Instruction: The new programming instruction "SEND SM—Send a Status Message from a Buffer" on page 293 has been added.

Contents

Figures

Using the Communications Facility

This chapter explains the Communications Facility functions you can use in writing applications and device-support (I/O control) programs. It summarizes what you can do with the Communications Facility and what instructions you use to carry out each function.

Language Extension Instructions

Your program will communicate with the Communications Facility through a set of EDL instructions collectively called *language extensions*. The instructions fall into these categories:

- *Storage management* instructions, which enable you to get and free blocks of processor storage for your program's use. You'll be using these instructions to get the storage you need to build, send, and receive messages.
- *Message management* instructions, which enable you to send messages to and receive messages from stations.
- *Station management* instructions, which allow you to create and delete station blocks and gain access to information about existing stations.
- *3270 field formatting* instructions, which allow you to create and receive data as a 3270 data stream.
- *Data move* instructions, which allow you to carry out indexed and indirect moves.
- A *system facilities* instruction, which allows you to gain access to various system control blocks.
- *Task management* instructions, which allow you to create task control blocks and activate and deactivate tasks.
- *Queue management* instructions, which allow you to define and access queues.
- *Supervisor definition* instructions, used to define elements of a Communications Facility configuration to the Event Driven Executive (EDX) supervisor. These instructions are used only for system generation; they cannot be used in your application program.
- *Local Communications Controller* instructions, used to access the Local Communications Controller. These instructions are for Communications Facility internal use only and are described in the *Debugging Guide.*

The following list summarizes these instructions and shows where you can locate them in this book.

Storage Management

"DEFINE BUFFER—Define a Buffer" on page 215

"DEFINE BUFFERPOOL—Define a Workspace Pool" on page 217

“FREE B—Free a Buffer” on page 195

“FREE S—Free Storage” on page 197

“GET B—Get a Buffer” on page 203

“GET S—Get Storage” on page 213

“GET W—Get Workspace” on page 215

Message Management

“RECEIVE M—Receive a Message into a Buffer” on page 259

“RECEIVE N—Receive Notification of Messages” on page 263

“RECEIVE P—Purge a Message” on page 265

“RECEIVE T—Receive a Message into a Text Area” on page 267

“SEND A—Send Acknowledgment” on page 271

“SEND CP—Send a Command” on page 273

“SEND E—Send an Error Message” on page 277

“SEND L—Send a Log Message” on page 279

“SEND M—Send a Message from a Buffer” on page 281

“SEND MT—Send a Transaction from a Buffer” on page 285

“SEND S—Send a Status Message from an EDX Text Area” on page 289

“SEND SM—Send a Status Message from a Buffer” on page 293

“SEND T—Send a Message from an EDX Text Area” on page 297

“SEND TT—Send a Transaction from an EDX Text Area” on page 301

Station Management

“LOCATE NA—Locate a Station Block by Network Address” on page 219

“LOCATE ST—Create, Delete, Purge, or Locate a Station Block” on page 221

3270 Field Formatting

"GET AI—Retrieve the AID Byte and Cursor Address from a 3270 Data Stream" on page 201

"GET F—Retrieve a Field from a Buffer" on page 205

"PUT AID—Put an AID Byte into a 3270 Data Stream" on page 227

"PUT CO—Put a WRITE Command into a 3270 Data Stream" on page 229

"PUT CURS—Put a Cursor into a 3270 Data Stream" on page 231

"PUT DLEETB—Put a DLE and an ETB into a 3270 Data Stream" on page 233

"PUT DLEETX—Put a DLE and an ETX into a 3270 Data Stream" on page 235

"PUT DLESTX—Put a DLE and an STX into a 3270 Data Stream" on page 237

"PUT ERA—Put an Erase Order into a 3270 Data Stream" on page 239

"PUT ETB—Put an ETB into a 3270 Data Stream" on page 241

"PUT ETX—Put an ETX into a 3270 Data Stream" on page 243

"PUT F—Put Data into a Communications Facility Buffer" on page 245

"PUT NUL—Set Buffer Address" on page 249

"PUT REP—Repeat a Character in a 3270 Data Stream" on page 253

"PUT STX—Put an STX into a 3270 Data Stream" on page 255

Data Move

"GET F—Retrieve a Field from a Buffer" on page 205

"PUT F—Put Data into a Communications Facility Buffer" on page 245

"MOV—Move Data" on page 225

Facilities

"GET A—Locate a System Facility" on page 199

Task Management

"ACTIVATE T—Activate or Deactivate a Task" on page 179

"PUT TCB—Create a Task Control Block" on page 257

Queue Management

"DEFINE Q—Define a Queue Control Block" on page 191

"GET Q—Get an Element from a Queue" on page 211

"PUT Q—Put an Element in a Queue" on page 251

Supervisor Definition

"DEFINE BRB—Define a Buffer Reference Block" on page 183

"DEFINE DEVICE—Define Remote Disk Access" on page 189

"DEFINE VOLUME—Define a Remote Disk Volume" on page 193

"CFTERM—Define Non-EDX 3101 or 7485 on Multifunction Attachment" on page 181

"LCC—Define Local Communications Controller Channel" on page 217.

Communications Facility Messages

Much of the work your Communications Facility programs will do consists of building, sending, receiving, and otherwise managing *messages*. A message is a unit of data to be transmitted from one station to another. It may, for example, be a request that a terminal operator answer a question; the operator's response; a report to be produced on a printer; a Communications Facility operator command; or a very large data set. A message can be of any length up to 32K bytes.

The Communications Facility language extensions include instructions to help you build, send, and receive messages.

Message Type

There are five types of Communications Facility messages:

- *Data*. The message is intended to be sent to another station. The data in the message is meaningful to its destination; the Communications Facility doesn't have to understand it.

- *Transaction*. The message is a Communications Facility transaction to be processed at some Series/1 in the configuration. A transaction is always sent to the program dispatcher, which then routes it to its ultimate destination. The first part of the message, the transaction header, tells the program dispatcher where to route the transaction.

- *Command.* The message is a Communications Facility command. Using this type of message allows your program to issue commands.
- *Log.* The message is an error message or an informational message to be sent to the system log device. By creating this kind of message, your program can log its error messages just as the Communications Facility logs its own.
- *Status.* The message contains control information, such as an X.25 control message, or a command that tells a program to stop.

Your program may create and send messages of any type, and may receive data, transaction, or status messages.

Managing Storage

You need storage to receive messages and to create the messages you send. You may also need storage for other purposes. There are three types of storage areas:

1. *Communications Facility buffer*—An area up to 32K bytes long prefixed by a 5-word header, which is used to keep track of the data in the buffer, as explained in section "Using the Buffer Header" on page 8.
2. *EDX text area*—An area up to 254 bytes long prefixed by a 2-byte text count field, which contains the size of the area and the count of the actual number of characters in the area.
3. *Workspace*—An area up to 32K bytes long with no header or other control information.

If you're creating or receiving a data, transaction, or status message, the space can be either an EDX text area or a Communications Facility buffer.

If you're creating a command message, you must build the message in an EDX text area.

If you're creating a log message, you can prestore the message text in a member of the $.SYSMSG data set. A log message you send can consist of prestored text alone; prestored text with appended data from an EDX text area; or the contents of an EDX text area alone.

The storage you use for purposes other than creating or receiving messages can be a Communications Facility buffer, an EDX text area, or a workspace.

Using Workspace Pools

The storage you use in your program may be a *workspace pool*, an area that includes information used to control the allocation of space from a pool. A pool starts with a 4-word *buffer reference block* (BRB):

Word 1	Address of the first element in the pool.
Word 2	Address of the first element in the pool (initialized to zero).
Word 3	Size of the largest possible element (initialized to zero).
Word 4	Size of the pool, excluding the BRB.

Each element in the pool starts with a 4-word *storage resource block* (SRB):

Word 1	Address of next element in pool.
Word 2	Address of previous element in pool.
Word 3	Address of owner's task control block (zero if element is available space).
Word 4	Size of element excluding SRB.

When you build a pool, as explained in section "Using Dynamic Storage," you must create the BRB. Otherwise, you don't need to be aware of the BRB or the SRBs except to allow space for them when you compute the size of a pool.

The size of a workspace pool is the sum of the sizes of the workspaces and buffers to be allocated at the same time plus the following overhead:

- Add 8 bytes for the BRB at the beginning of the pool.
- For each workspace allocated from the pool, add 8 bytes for the SRB and round the sum to a multiple of 8.
- For each buffer allocated from the pool, add 8 bytes for the SRB and 10 bytes for a buffer header, and round the sum to a multiple of 8.

Assume, for example, that you need a pool large enough for two 30-byte workspaces and two 500-byte buffers. The space required for each workspace is 40 bytes (30 plus 8, rounded to a multiple of 8). The space required for each buffer is 520 bytes (500 plus 18, rounded to a multiple of 8). The required pool size is 1128 bytes (2 times 40, plus 2 times 520, plus 8).

Defining Pools in Your Program

You can define a pool in your program in two ways: by building one in dynamic storage or by using the DEFINE BUFFERPOOL instruction.

Using Dynamic Storage

You can use dynamic storage, specified on the PROGRAM statement, as a workspace pool. This approach has the advantage that you can use the SS function of the $DISKUT2 utility to change the size of the pool without recompiling your program. If you choose to use dynamic storage as a workspace pool, you must format the first 4 words of the area as a BRB. The code to do that is shown in Figure 1 on page 7.

Using DEFINE BUFFERPOOL

The DEFINE BUFFERPOOL instruction allows you to define a workspace pool within your program. When you use DEFINE BUFFERPOOL, you don't have to create the BRB, and you can change the size of the area only by recompiling the program. When you issue DEFINE BUFFERPOOL, you specify how big the pool is to be.

```
         IF ($LENGTH,LT,1)                        NO STORAGE ALLOCATED
           SEND ERROR,...                         LOG ERROR
           PROGSTOP LOGMSG=NO                     TERMINATE EXECUTION
         ENDIF
         MOVE #1,$STORAGE                         GET ADDRESS OF DYNAMIC
         MOVE POOL@,$STORAGE                      STORAGE AND SAVE IT
*                                                 BUILD BUFFER REFERENCE BLOCK
         ADD POOL@,+PHDRL,RESULT=(0,#1)           ADDRESS OF FIRST ELEMENT
         MOVE(2,#1),0,2                           TWO WORDS OF ZERO
         SHIFTL $LENGTH,8,RESULT=(6,#1)           STORAGE SIZE (BYTES)
         SUB(6,#1),+PHDRL                         LESS HEADER SIZE
         •
         •
         •
POOL@    DATA F'0'                                ADDRESS OF WORKSPACE POOL
PHDRL    EQU 8                                    LENGTH OF BUFFER REFERENCE BLOCK
```

Figure 1. Building a Buffer Reference Block

Getting Workspace from Your Program

The GET W instruction allows you to get some space from a workspace pool in your program, either the one built in dynamic storage or one defined through a DEFINE BUFFERPOOL instruction. When you issue GET W, you specify which pool the space is to come from; how large the space is to be; and whether your program is to wait, if necessary, until the space is available. When you've finished with the space, use the FREE S (free storage) instruction to return it to the pool.

You might, for example, use GET W to get space to save information about a station with which your program communicates.

Getting a Buffer from Your Program

The DEFINE BUFFER instruction defines a Communications Facility buffer within your program. The buffer is static; you can't release the space it occupies when your program is running. When you issue DEFINE BUFFER, you specify the buffer's size and, if you want, a character with which the buffer is to be filled initially (the default is a blank).

The GET B instruction gets a Communications Facility buffer from a workspace pool in your program, either the one built in dynamic storage or one defined through a DEFINE BUFFERPOOL instruction. When you issue GET B, you specify the pool the buffer is to come from, its size, and whether your program is to wait, if necessary, until the space is available. You can use the FREE B or the FREE S TYPE=BUFFER instruction to free a buffer obtained this way.

Getting Storage from the System Storage Pool

The GET S instruction lets you get a workspace or buffer from the system storage pool, S$POOL, in the common area. The Communications Facility uses S$POOL for station blocks and work areas. You should use S$POOL only for data that must be in partition 1 (for example, a device control block) or for data that is used by several programs in different partitions.

When you issue GET S, you specify how much space you need, whether it is to be formatted as a buffer, and whether your program is to wait, if necessary, until the space is available.

Use either the FREE B or the FREE S TYPE=BUFFER instruction to free a buffer in S$POOL. Use the FREE S instruction to free a workspace in S$POOL.

Using the Buffer Header

However you get your buffer, it is prefixed with a 5-word buffer header. Figure 2 on page 9 shows the buffer header contents.

The data you put into the buffer goes after the header. The address of the buffer or the name you give to it refers to the beginning of the data. You can send and receive messages without any awareness of the buffer header contents.

You may want to examine or change some of the buffer header fields. For example, you can find the length of a received message by examining B$COUNT (the second word of the buffer header). As another example, you might want to send the data at the end of a buffer, leaving out some data at the beginning of the buffer. You could do that by changing B$ADDR (the third word of the buffer header) to point to the data you want to send and B$COUNT (the second word of the buffer header) to the appropriate length.

There are three ways you can refer to the buffer header fields:

- By name (B$SIZE, B$COUNT, etc.) relative to the beginning of the buffer data. For example, you might use *buffername*+B$COUNT, or, assuming the address of the buffer is in register 2, (B$COUNT,#2). To use these names, you must include the Communications Facility system equates (S$CFEQU) in your program.

- By position relative to the beginning of the buffer data. For example, you can refer to B$COUNT as *buffername*-8.

- By giving names to the fields through the P1-P5 operands of DEFINE BUFFER.

Note that the fifth word of the header (B$TXTCT) is a text count field when the buffer size is less than 255 bytes. You can use such a buffer for instructions that require an EDX text area as well as for those that require a Communications Facility buffer.

Pn	Label	Displacement	Contents
P1[1]	B$SIZE	-10	Size, in bytes, of the entire buffer not including the header.
P2[1]	B$COUNT	-8	Count of the bytes of actual data in the buffer, beginning at B$ADDR.
P3[1]	B$ADDR	-6	The address of the first byte of the buffer to be treated as data.
P4[1]	B$DATA@	-4	An address of some data within the buffer (used when the buffer is being accessed by GET and PUT instructions).
P5[1]	B$TXTCT	-2	The EDX text count (maximum size followed by actual size of the data in the buffer) if the buffer is smaller than 255 bytes; the buffer size (equal to B$SIZE) if the buffer size is greater than 254 bytes.
	B$TEXT	0	Normally, the start of the data. B$ADDR normally points here.

Figure 2. Buffer Header Contents

Initializing Storage

When you get the first workspace or buffer from a workspace pool, the entire pool is initialized to binary zeros. When you get a workspace from a pool, the workspace is initialized to binary zeros. When you get a buffer from a pool, the buffer is not initialized. If you require that the buffer have some initial value, you must initialize it yourself.

Sending and Receiving Messages

The Communications Facility SEND and RECEIVE instructions allow your program to receive messages from one or many sources, process the messages, and send the results to one or many destinations.

A typical application program, shown in Figure 3 on page 10, may issue a RECEIVE instruction with an option that indicates the program is to wait until there is a message to be received on its queue. When a message is queued, the message text is then moved to an area specified on the RECEIVE instruction so the program can process the message.

After processing the message, the program may send it on to another program for further processing, or create and send some other output message. The output may go to the originator of the original message, or to any other station.

1 For use in DEFINE BUFFER

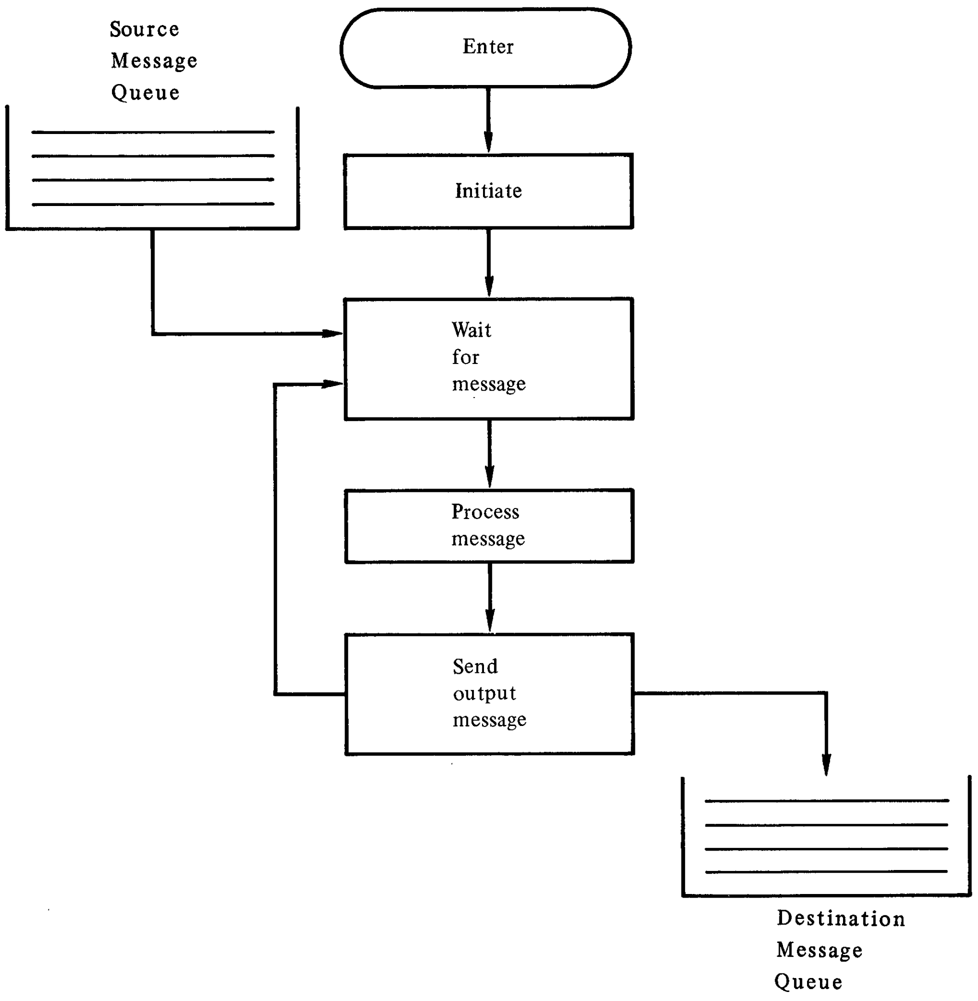


Figure 3. Sample Application Program Flow

Sending Messages

This section explains the instructions that allow you to send data, transaction, command, and status messages. The instructions are:

SEND M	Send a data message from a Communications Facility buffer.
SEND T	Send a data message from an EDX text area.
SEND MT	Send a transaction from a Communications Facility buffer.
SEND TT	Send a transaction from an EDX text area.
SEND CP	Send a command from an EDX text area.
SEND SM	Send a status message from a Communications Facility buffer.
SEND S	Send a status message from an EDX text area.

Message Content

When you send a message, you specify the address of the buffer or text area that contains the message. The buffer header field B$COUNT or the text count field must contain the message length.

A data message can be any data that is meaningful to the program that receives the message.

A command message can be any of the CP or PD commands described in the *Operator's Guide.*

By convention, status messages are used only for sending certain types of control information. The command processor and the program dispatcher send status messages to tell programs to stop or halt. The X.25 I/O control program uses status messages to send X.25 control messages. You should send status messages from your application only to send X.25 control messages, as discussed in the chapter "Writing an X.25 Application Program" on page 107.

The first part of a transaction message is a fixed-format header that identifies the transaction and the cell where it is to be processed.

The format of a transaction message is:

1-4	5-6	7-10	11-12	13-*n*
tid1	*c1*	*tid2*	*c2*	*transaction data*

where:

tid1
: is the primary transaction identifier, a 4-character code that identifies the transaction.

c1
: is the primary cell identifier, a 2-character code that identifies the cell where the transaction is to be processed. Two blanks or 00 means that the transaction is to be processed in the cell where it originated. Two asterisks mean that the transaction is a broadcast transaction, which is to be processed in all cells known to the program dispatcher.

tid2
: is the secondary transaction identifier, whose meaning is defined by the program that processes the transaction.

c2
: is the secondary cell identifier, whose meaning is defined by the program that processes the transaction. When *c2* is ??, the program dispatcher replaces it with the ID of the cell where the transaction originated.

transaction data
: is whatever data is meaningful to the program that processes the transaction.

The minimum length of a transaction message is 6 bytes. The data portion is optional. If a transaction contains no data, the secondary transaction identifier and secondary cell identifier are optional.

Message Destination

The destination of data and status messages is a station. You can specify the name of the destination station in a SEND instruction. You can also specify an area that will contain the name when the program is executed. This allows you to obtain the name of a station from which you receive a message and use that name as a destination when you send a response.

If you don't specify the destination of a data or status message, the message is sent to the origin station's *direct link*—the station specified as the default destination by the CP LINK command. The direct link can be changed at any time, even while your program is running.

The destination of command messages is the command processor; either in the local node or in a remote node. If you want a command to be processed in the local node, you don't need to specify a destination in the SEND CP instruction. If you want a command to be processed in a remote node, you must specify the name of any station in that node as the destination. This specification causes the command to be routed to the remote node, where it is sent to the command processor.

You can't send a command message to a remote node if the nodes are connected by a BSC multipoint line (one managed by I/O control programs $.IO0AC0 and $.IO0AE0).

The initial destination of transaction messages is the program dispatcher (that is, the station named $.PD). You don't specify a destination in SEND MT and SEND TT instructions, because $.PD is the implied destination. The ultimate destination is the transaction-processing program defined by the primary transaction and cell identifiers in the transaction header.

Message Origin

The origin of a message is a station. You can specify the name of the origin station in a SEND instruction. If you don't specify the name, the origin is the station with the same name as the sending program.

If the origin station doesn't exist, the message origin is not known. This is not necessarily a problem; however, the message origin is required when:

- You haven't specified the destination of a data or status message, and the message is sent to the origin station's direct link. If the origin isn't known, then the destination can't be determined and the message is undeliverable.

- You request that your program wait until the message has been received, as described in "Waiting for Completion" on page 13. If the origin isn't known, the request is ignored.

- A program that receives the message needs the name of the origin station.

Message Priority

When you send a message, you can assign it a priority of 1 (highest priority) to 127 (lowest priority). The default is 127. Messages for a given destination are queued and received according to their priority.

At your installation's option, messages of priority 127 for a given station can be queued on disk or diskette while all its other messages are queued in processor storage. In general, disk-queued messages take longer to arrive. As a rule, you should disk-queue longer messages and messages for which arrival time isn't crucial.

Waiting for Storage

When you send a message, it is copied into the message buffer pool (CFBUF). A disk-queued message remains in CFBUF until it has been written to disk. A storage-queued message remains in CFBUF until it has been received.

The WAIT option of the SEND instructions allows you to specify whether you want your program to wait, if necessary, until there is space in the message buffer pool for the message. The default is YES.

Waiting for Completion

There are three different points at which a SEND instruction will complete (that is, when your program is free to resume execution), depending on the operands you specify:

- If you specify ACK=NO (which is the default), your program is free to resume execution when the message has been placed on the destination queue. If the message is disk-queued, the SEND instruction does not complete until the message has been written to disk.
- If you specify ACK=YES, your program is free to resume execution when the message has been received, as explained in the section "Acknowledging Messages" on page 17. ACK=YES has no effect if the message origin is not known, if the origin is a vector station (one that represents a station in a remote node), or if the destination is a message station.
- If you specify OPTION=NOPOST for a disk-queued message, your program is free to resume execution when the message has been placed on the disk write task's queue. (The effect of OPTION=NOPOST is the same as ACK=NO for a storage-queued message.) This option is provided for the Communications Facility log processor, and it is not recommended that you use it in your programs. If you do use it, the SEND instruction completes successfully even when a disk-queued message can't be delivered.

Undeliverable Messages

A message may be undeliverable—for example, if its destination station is stopped or doesn't exist. At your installation's option, such messages are either discarded or queued to a station called $.WASTE. Each undeliverable message on $.WASTE's queue is preceded by a reason message, which contains the date, time, and a code that indicates why the message could not be delivered. You can use utility program $.UT2 (described in the *Operator's Guide*) to examine the messages on $.WASTE's queue.

If you write a program that receives messages and then sends them on, you may decide that a message is undeliverable. It might, for example, contain incorrect data. If you want to send such a message to $.WASTE, use a SEND instruction with OPTION=WASTE. Don't specify $.WASTE as the destination station instead. If you do, the message will be sent to $.WASTE, but it won't be preceded by a reason message.

When you use OPTION=WASTE, you don't specify a reason code. The code will always be WA07, which indicates that the message was sent to $.WASTE by some program other than the message dispatcher.

When you send a message that can't be delivered, you may not want it to be sent to $.WASTE. For example, if the destination is stopped, you may send the message to some other station. To prevent an undeliverable message from being sent to $.WASTE, specify OPTION=DISCARD in the SEND instruction.

Receiving Messages

The RECEIVE M and RECEIVE T instructions allow you to receive a data, transaction, or status message. You use RECEIVE M to receive the message into a Communications Facility buffer and RECEIVE T to receive it into an EDX text area.

A return code of +6 on completion of the RECEIVE instruction means that the message is a status message. Any program can receive a status message that is a command to stop or halt, as discussed in the section "Terminating Your Program" on page 28. X.25 application programs may also receive status messages that contain X.25 control messages, as discussed in "Writing an X.25 Application Program" on page 107.

A return code of -1 on completion of the RECEIVE instruction means that the message is a data or transaction message. There's no way to distinguish a data message from a transaction message once you've received it; your program must know what kind of messages it's going to receive.

You receive messages from a station's message queue. The station can be the one that represents your program or some other station. To receive messages from some other station's queue, you must specify the name of the station. You can specify that, if there is no message on the queue, the instruction is to wait until there is a message on the queue.

You must specify the address of the buffer or text area into which the message is to be placed. On successful completion, the message length is in the buffer header field B$COUNT or in the text header field.

You can specify whether or not receipt of the message is to be acknowledged to the sender (described under "Acknowledging Messages" on page 17). You can also request the name of the station that originated the message, or you can request the message header (described under "Examining or Sending the Message Header" on page 15).

If you request the name of the originating station, you provide a 10-byte area for the station name. After the receive, the first 8 bytes of that area contain the station's name. The remaining 2 bytes contain the station's type and subtype, as shown in Figure 4 on page 15. S$CFEQU contains names for the type and subtype values. There are types and subtypes other than those shown in the figure; the figure presents those from which you are most likely to receive messages. All the types and subtypes are shown in the *Operator's Guide.*

Station	Type	Subtype
User station	02	00
3277 terminal	04	C2
Emulated channel attach port	04	D0
Emulated 3277 terminal	04	E2
3101 device, managed as a 3277	06	31
4978 device, managed as a 3277	06	78
3101F device, managed as a 3277	06	F3
Message station	0C	00
SNA LU 3277	12	E2
SNA LU 3278	12	EA
SNA LU 3279	12	EB
PVC	16	BD
SVC	16	BE

Figure 4. Station Types and Subtypes

When you receive a message from a queue, the message is removed from the queue unless you specify otherwise. The COPY option of RECEIVE M and RECEIVE T allows you to receive a copy of the message but also leave it on the queue. When you have finished processing the message, you must remove it from the queue, as explained later in this section. Until you do, another RECEIVE with the COPY option will receive another copy of the same message, unless a higher-priority message has been placed on the queue in the meantime. Your program is responsible for being sure that the next RECEIVE is receiving the right message.

The KEEP option is just like the COPY option except that, if the message is disk-queued and the receiving program's buffer is too small to receive the message, a copy is kept in the message buffer pool (CFBUF) as well as on the disk queue. It is unlikely that you will have any reason to use the KEEP option in your programs. The option is provided for use of the Series/1-to-Series/1 I/O control program.

A third RECEIVE instruction, RECEIVE P, purges the last message received with the COPY or KEEP option. After your program issues a RECEIVE with the COPY or KEEP option, it can't issue a RECEIVE without the option until it issues a RECEIVE P.

Note that only one program at a time should use RECEIVE with the COPY or KEEP option against a queue; multiple programs would interfere with each other's operations on the queue.

A fourth RECEIVE instruction, RECEIVE N, merely notifies you of whether there are any messages on the queue, without receiving one. When you issue RECEIVE N, you can specify whether the instruction is to wait until there is a message queued.

Examining or Sending the Message Header

When the Communications Facility transmits a message, it prefixes a 24-byte *message header* to the message. The header contains such information as the message's origin, destination, and priority. You can send and receive messages without being aware of the header at all.

If you want to examine the header, you can specify, on the RECEIVE M or RECEIVE T instruction, a location where you want the Communications Facility to place the header. The header's contents are shown in Figure 5.

Having received the header, you can refer to its fields in two ways:

- By name (MNIQ, MPIQ, etc.) relative to the beginning of the header. To use these names, you must include the Communications Facility system equates (S$CFEQU) in your program.

- By displacement from the beginning of the header.

When you use a SEND M instruction to send a data message or a SEND CP instruction to send a command, you can provide the message header as well as the message data. This option is provided for use of the I/O control programs that transfer messages and their headers between Series/1s. It is unlikely that you will have any reason to use it, except as described in the section "Intercepting Messages from an I/O Control Program" on page 37.

If you do provide headers for the messages you send, you can specify any value for fields MNIQ, MPIQ, M$SAKR, and M$STA; the message dispatcher provides this data. You must provide the rest of the message header data.

Label	Size	Displacement	Contents
M$NIQ	2	0	The address of the next message for this station. If this element is 0, this is the last message on the station's storage queue.
M$PIQ	2	2	The address of the previous message for this station. If this element is 0, this is the first message on the station's storage queue.
M$PRI	1	4	The priority of the message.
M$SAKR	1	5	The sender's address key register (AKR) value.
M$STA	2	6	The sender's task control block (TCB) address.
M$ALV	2	8	The network address of the origin station's alternate link; 0 if it has no alternate link.
M$FLAG	2	10	Message option flags. See the *Debugging Guide.*
M$OAF	2	12	The origin station's network address; 0 if the origin station is not known.

Figure 5 (Part 1 of 2). Message Header Contents

Label	Size	Displacement	Contents
M$DAF	2	14	The destination station's network address.
M$SNF	2	16	A binary number representing the sequence number of the message, assigned at its origin.
M$DCF	2	18	A binary number representing the number of bytes in the message.
M$RH	2	20	Message type flags. See the *Debugging Guide*.
M$DATA@	2	22	Always 0.
M$TEXT	—	24	The beginning of the message data.

Figure 5 (Part 2 of 2). Message Header Contents

Acknowledging Messages

When you send a message, you can use the ACK operand of the SEND instruction to specify that your program is to wait until the message has been received.

The receiver acknowledges receipt of the message either:

- By specifying ACK=YES on the RECEIVE M or RECEIVE T instruction

or

- By specifying ACK=NO on the RECEIVE instruction and later issuing another instruction, SEND A, which only acknowledges receipt.

To use SEND A or ACK=YES, the sender and receiver must be in the same node. If they are in different nodes, the sender's wait is satisfied when the I/O control program that will transfer the message to the remote node receives the message.

Using ACK=YES on a SEND instruction keeps two programs operating synchronously. It paces the flow of messages through the system and, when the messages are queued in storage, keeps CFBUF from being flooded.

Sending Program	Receiving Program
1. SEND with ACK=YES	
	2. RECEIVE with ACK=YES
3. Wait is posted	
	4. Process message

If the receiving program uses SEND A instead of RECEIVE with ACK=YES, the sending program can wait until the receiving program has not only received the message, but also processed it.

Sending Program	Receiving Program
1. SEND with ACK=YES	
	2. RECEIVE with ACK=NO
	3. Process message
	4. SEND A
5. Wait is posted	

If you're sending messages between Series/1s or between a Series/1 and a host computer, SEND A and RECEIVE with ACK=YES are not effective. To acknowledge messages, the communicating programs will have to have an agreed-on protocol. For example, after receiving a message (or after receiving and processing a message), each may send the other a special acknowledgment message.

Sending Program	Receiving Program
1. SEND data message	
	2. RECEIVE data message
	3. Process data message
	4. SEND acknowledgment message
5. RECEIVE acknowledgment message	

When you send transactions, the only effect of ACK=YES is to synchronize the operations of the sending program and the program dispatcher. The sender's wait is satisfied when the program dispatcher receives the transaction. The situation is the same as the first example in this section, with the program dispatcher as the receiving program.

You can synchronize the operations of transaction-processing programs by sending a transaction to acknowledge that a transaction has been received and processed. The sending program will wait to receive the acknowledgment transaction, so there's no point to specifying ACK=YES on the SEND instruction.

Sending Program	Program Dispatcher	Receiving Program
1. SEND transaction		
	2. RECEIVE with ACK=YES	
	3. SEND transaction	
		4. RECEIVE transaction
		5. Process transaction
		6. SEND acknowledgment transaction

Sending Program	Program Dispatcher	Receiving Program
	7. RECEIVE with ACK=YES	
	8. SEND acknowledgement transaction	
9.RECEIVE acknowledgement transaction		

Creating and Sending Log Messages

By using the $.CONFIG utility and the SEND E or SEND L instruction, you can build and send your own log messages just as the Communications Facility handles its own log messages.

Creating Messages in $.SYSMSG

If you want, you can store the text portion of your log messages in the same data set the Communications Facility uses for its own log messages. The data set, $.SYSMSG, is a partitioned data set. Each member is a file that accommodates up to 99 messages. To create a member for your messages and store your message text, or to edit a member you've already created, use the EDIT command of the $.CONFIG utility program.

Note that the Communications Facility itself uses eight file IDs: CA, CF, CP, IO, I1, PD, PN, and SN. Give your file a name different from any of those. When you send a log message, the default file ID is UM. You may want to name your file UM so you can use that default. $.CONFIG creates a file, with the ID you specified, containing 99 messages that all have UNDEFINED MESSAGE as their text.

See the *Operator's Guide* for instructions on how to use the $.CONFIG utility program.

Sending Log Messages

This section explains the instructions that allow you to send a log message, and shows the format of the delivered message.

SEND L and SEND E Instructions

The Communications Facility includes two instructions, SEND L and SEND E, that allow you to send a message to the Communications Facility system log.

You can specify a message identifier when you send a log message; the default is UM. If you want the message to include text from $.SYSMSG, the identifier must be the name of the member that contains the text.

You must specify a message number, a value from 1 to 99. The number appears in the log message and is used to locate the message text in the $.SYSNET member.

You can specify message text in the instruction itself. If you don't use $.SYSMSG for your log messages, the text can be a complete message. If you do use $.SYSMSG, the text can provide additional, variable information. You might, for example, have text in $.SYSMSG that describes an error and append to it the name of the station to which the error applies. The text you specify in the instruction plus the text taken from $.SYSMSG must not exceed 48 characters. If it does, the excess text is truncated.

Other options of SEND L and SEND E allow you to:

- Create a copy of the message in a text area.
- Put a 4-character hexadecimal code into the message. You might use this field, for example, to display a return code.
- Specify the type code: E for error, I for information, and C for comment. SEND E has two special type codes: X, which is used only in task error exit routines to log special data; and D, which produces a diagnostic dump. See "Using Task Error Exits" on page 29 and "Using Diagnostic Dumps" on page 29 for an explanation of these special cases.

The only difference between SEND L and SEND E is the default type code: I for SEND L, and E for SEND E. A type code of C (comment) allows you to create a copy of the message in a text area without sending it to the system log.

When your program issues a SEND L or SEND E instruction, it does not resume execution until the log message has been delivered to the system log. If the system log is an EDX device and the device is busy, your program waits until the device is available. If the system log is a Communications Facility station, your program waits until the log message has been put on that station's message queue.

Format of the Delivered Message

A log message, as delivered, has this format:

phh:mm:ss idnn t code program apptext dstext

where:

p
: is a 1-character prefix: * for an error message, blank for an information message.

hh:mm:ss
: is the time of day (taken from the EDX time of day clock) when the message was logged.

idnn
: is the message identifier followed by the message number.

t
: is the type code: E for error, I for information.

code
: is the 4-digit hexadecimal code you specified when you sent the message. 0000 means you didn't specify a code.

program
: is the name of the program that issued the message.

apptext
is the text you specified in the instruction.

dstext
is the text from $.SYSMSG.

Managing Stations

The Communications Facility includes three instructions that allow you to control stations from your program.

Sending Commands

The SEND CP instruction allows your program to send a command, as described in the section "Sending Messages" on page 10. You might use SEND CP, for example, to start, stop, or link a station, or to modify a station's attributes.

Creating, Purging, Deleting, and Examining Station Blocks

Two instructions, LOCATE ST and LOCATE NA, allow you to perform various operations on station blocks.

LOCATE ST allows you to:

- Obtain the address of a station's control block. You can then examine or change the control block contents. The control block formats are given in the *Debugging Guide.*

- Create a station block. A station created in this way can't have a disk queue and need not be defined in $.SYSNET. Normally, you would use this option only to create a message station for temporary use within your program or to create a station block for your program when it is started by an EDX $L command rather than by a CP Start command.

  If you use LOCATE ST to create a station block, you're responsible for deleting that station block and thus freeing the space it occupies.

- Delete a station. This option deletes a station block if the station has no storage-queued messages.

- Purge a station. This option removes all storage-queued messages from a station's queue and deletes its station block.

- Stop your program. This option removes all storage-queued messages from the station's queue, deletes its station block, and performs a PROGSTOP.

You should delete only stations that you manage—the one that represents your program and those that you create. If you delete stations managed by I/O control programs or other parts of the Communications Facility, the results are unpredictable.

The LOCATE NA instruction returns the address of a station's control block, given the network address of the station. LOCATE NA has no other options.

Creating and Retrieving 3270 Data Streams

The language extension instructions include 13 PUT instructions and two GET instructions to help you in creating and retrieving 3270 data streams. Information about 3270 data streams and their contents is in the *3270 Description and Programmer's Guide.*

Creating 3270 Data Streams

The 13 PUT instructions move data, plus 3270 commands, orders, and control information, from an EDX text area to a Communications Facility buffer. The PUT instructions use the B$COUNT and B$DATA@ fields of the buffer header to keep track of the next available buffer location. Thus you can use multiple PUT instructions to construct a 3270 data stream field by field.

The PUT F instruction moves data or control information from a text area to a buffer. You can use PUT F to move any sort of data—not necessarily a 3270 data stream—to a buffer. When you use PUT F, you must specify where the data is and the buffer to which it is to be moved. You may, optionally, specify that the field is to be the first or the last in the buffer.

When you use PUT F to move 3270 field data, you must specify where the data is to be displayed on the output device; you may, optionally, specify a field attribute character and put a TAB order into the buffer following the data.

The remaining 12 PUT instructions move 3270 commands, orders, and BSC control information from a text area to a buffer. You could use PUT F to move data of this sort, but, because the data is non-character, it is easier to use the other PUT instructions.

They are:

PUT AID—Moves a 3270 read header to the start of the buffer. You supply the attention ID value and, optionally, a cursor position.

PUT CO—Moves a 3270 write command sequence to the start of the buffer. The data includes a WRITE command and a write control character that resets modified data tags and unlocks the keyboard. At your option, you can change the command to an ERASE WRITE or an ERASE/WRITE ALTERNATE, and you can change the write control character.

PUT CURS—Appends a 3270 insert cursor order to the data stream, with the cursor address you specify.

PUT DLEETB—Appends a DLE and an ETB to the data stream, indicating the end of a block of transparent data.

PUT DLEETX—Appends a DLE and an ETX to the data stream, indicating the end of transparent text.

PUT DLESTX—Moves a DLE and an STX to the start of the buffer, indicating the start of transparent text.

PUT ERA—Appends a 3270 erase unprotected to address order to the data stream, with the stop address you specify.

PUT ETB—Appends an ETB to the data stream, indicating the end of a block of text.

PUT ETX—Appends an ETX to the data stream, indicating the end of text.

PUT NUL—Appends a 3270 set buffer address order to the data stream, with the address you specify.

PUT REP—Appends a 3270 repeat to address order to the data stream, with the repeat character and stop address you specify.

PUT STX—Moves an STX to the start of the buffer, indicating the start of text.

Retrieving 3270 Data Streams

The GET F instruction moves data from a Communications Facility buffer to an EDX text area. The maximum amount of data you can move depends on the size of the text area. After the move, you can find the number of bytes moved in the text area header.

By combining various operands of GET F, you can achieve four different kinds of retrieval: get sequential, get sequential by delimiter, get specific 3270 field, and get sequential 3270 field. For sequential retrievals, the GET F instruction uses the B$DATA@ field of the buffer header to keep track of the current buffer position.

Get Sequential: This form of retrieval gets the next field, starting at the current buffer position and retrieving the number of bytes indicated by your text area header. At completion, B$DATA@ points to the byte following the last one moved. You achieve this kind of retrieval by coding only the *text* and *buffer* operands. You can use this form of retrieval to retrieve any kind of data—not necessarily from a 3270 data stream.

Get Sequential by Delimiter: This form of retrieval moves the next field, from the current buffer position to a delimiter that you specify in the COMPARE= operand. At completion, B$DATA@ points to the byte following the delimiter. You can use this form of retrieval to retrieve any kind of data—not necessarily from a 3270 data stream.

Get Specific 3270 Field: This form of retrieval moves a field, whose screen position you specify, from a 3270 data stream. The Communications Facility searches the buffer for a set buffer address (SBA) order for the specified position. All data from that SBA to the next SBA, ETX, or ETB is moved. If you want, you can also specify that the field's attribute character be moved. You can intermix this form of retrieval with sequential retrievals; it does not affect B$DATA@.

Get Sequential 3270 Field: This form of retrieval moves the next field, as delimited by an SBA order, from a 3270 data stream. It also returns the field's screen position to you. If you want, you can also specify that the field's attribute character be moved. If the first field does not begin with an SBA order, its screen position is assumed to be zero (row 1, column 1).

The GET AI instruction retrieves the attention ID (AID) byte from a 3270 data stream. You can also use this instruction to get the cursor address or to verify the cursor's location. It is assumed that the data stream is the result of a 3270 read command.

Moving Data

The MOV instruction moves data from one storage location to another. You can use language extension symbolic addresses, indexed operands, and indirect operands for the data addresses and length.

The GET F instruction, described in "Retrieving 3270 Data Streams" on page 23, moves data from a Communications Facility buffer to an EDX text area. The buffer data doesn't have to be a 3270 data stream.

The PUT F instruction, described in "Creating 3270 Data Streams" on page 22, moves data from an EDX text area to a Communications Facility buffer. The buffer data doesn't have to be a 3270 data stream.

Locating System Facilities

The GET A instruction allows you to get the addresses of various system control blocks and tables.

By using GET A, you can get the addresses of these system facilities:

- Task control block
- Terminal control block
- The buffer within the terminal control block
- Current station block
- System storage pool (S$POOL)
- System queue control block
- EDX system common area ($SYSCOM)
- Language extension command table

Some Communications Facility programs refer to other system facilities by their negative displacement from the command table. These facilities are defined in module S$CSXSYS.

Managing Tasks

Activating and Deactivating Tasks

If you're writing an I/O control program with a reentrant task that controls multiple devices, you can use the ACTIVATE T instruction to activate and deactivate tasks. (You can't use EDX instructions ATTACH and DETACH to do this, because the task isn't defined by a TASK statement.) Options of ACTIVATE T allow you to attach and chain, unchain, or detach and unchain a task.

Creating Task Control Blocks

If you're writing an I/O control program with a re-entrant task that controls multiple devices, you need a task control block (TCB) for each station that represents one of the devices. The TCB can be within the station block in the system storage pool (S$POOL) or it can be in the I/O control program's storage. When the TCB is within the station block, it is created when the station is started.

You can use the PUT TCB instruction to create a TCB that is separate from, but associated with, a station. You specify the address of the station block and the address of a 130-byte area to contain the TCB. The PUT TCB instruction initializes the first 128 bytes of the area as a TCB, assigning a task priority according to the station type. The priority is 100 (level 2, priority 100) for a line station and 355 (level 3, priority 100) for any other type of station. The PUT TCB instruction places the address of the station block in the word following the TCB, and it places the address and address key of the TCB in the station block (in fields Q$TCB@ and Q$TCBADS).

A station that is associated with a TCB must have a station block long enough to contain the TCB address and address key. The location of these fields is defined in the Communications Facility system equates, S$CFEQU.

You can also use the PUT TCB instruction to create a TCB that is not associated with a station. You might do this, for example, when you need more than one task for a station. You specify the address of a 128-byte area to contain the TCB, and optionally, the task level and priority. The default level is 2, and the default priority is 150. When you omit the station specification, the PUT TCB instruction just initializes the 128-byte area as a TCB.

When you create a TCB that is not associated with a station, you can make the association yourself by placing the address of a station block in the word following the TCB. If you do this, you can use symbolic address #L and operand #LINE of the GET A instruction to address that station when the TCB is active.

Creating and Managing Queues

The Communications Facility uses three instructions—DEFINE Q, GET Q, and PUT Q—to create and manage its queues of messages. While these instructions are intended primarily for internal system use, they are available to you if you have a use for them.

DEFINE Q creates a 2-word queue control block (QCB). The first word contains the address of the first element in the queue, and the second word contains the address of the last element in the queue. Note that a Communications Facility QCB is different from an EDX QCB. An EDX QCB is a 5-word control block used to control access to a resource that can be used by only one task at a time.

GET Q gets an element from the queue. You can retrieve the first element in the queue, retrieve the first element of a specified priority, or remove an element from a queue.

PUT Q puts an element into the queue. The first 5 bytes of the queue element consist of the address of the next element in the queue (2 bytes), the address of the previous element in the queue (2 bytes), and the element's priority (1 byte).

Writing Communications Facility Programs

This chapter covers general programming topics that will be of interest no matter what type of program you're writing.

Copying the Communications Facility Equates

The Communications Facility equate table, S$CFEQU, contains equates for all the fields of the various station blocks, the buffer header, and the message header. If you want to be able to refer to any of those fields by name, include the statement 'COPY S$CFEQU' in your program.

Loading Your Program

Your program may be loaded in any of five ways:

- Through entry of a CP S (start) command that designates a user station defined in $.SYSNET that has the same name as your program.
- Through a transaction, if your program is defined to the program dispatcher as the program that services that transaction.
- Through the EDX $L command.
- Through a LOAD instruction issued by a user program.
- Through the $DEBUG program.

A CP S command creates a station block with the same name as your program. A transaction also creates a station block if you have defined the transaction appropriately. (The chapter "Creating a Transaction" on page 67 explains how to do that.)

If you use $L, $DEBUG, or a LOAD instruction to load your program, your program must include a setup routine that creates a station block for the program, gives the station a network address, and sets its status word to active.

The instructions to accomplish these steps, from a program named PROGA, are:

```
LOCATE   ST,#1,OPTION=CREATE            FIND OR CREATE STATION BLOCK
IF       (PROGA,EQ,-2)                  STATION BLOCK CREATED
  MOVE   (Q$NAU,#1),X'018A'             SET NETWORK ADDRESS
ENDIF
IOR      (Q$STAT,#1),+Q#ACTIVE          FLAG STATION ACTIVE
```

Verifying that a Partition is Mapped

The common area, which contains support for Communications Facility instructions, may not be mapped into all partitions. If a Communications Facility program is loaded into an unmapped partition, it will program check the first time a Communications Facility instruction is executed. (The program check PSW has a value of 0802 indicating an invalid function.)

Because of this restriction, the CP Start command processor and the program dispatcher will not load a Communications Facility program into an unmapped partition. If you load your program with an EDX $L command or LOAD instruction, and if your system has unmapped partitions, you may want to include code to verify that its partition is mapped. If your program loads another program, it should check before loading the secondary program. In either case, if the test fails, the program can terminate before executing any Communications Facility instructions.

Communications Facility initialization records partition mapping in a word in module CSXSYS, which resides in the common area. The address of the mapping word is in the EDX communications vector table, at label $CFPARM. (The label is defined in EDX copy code PROGEQU.)

This word is actually a bit map of the partitions: bits 0-7 correspond to partitions 1-8 and, therefore, to address keys 0-7. Bit value 0 means that the partition is mapped; 1 means that the partition is unmapped or doesn't exist. The second byte of the word (bits 8-15) is always set to X'00'. For example, a mapping word value of X'0300' means that partitions 1-6 are mapped.

The following instructions check whether or not the partition in which they execute is mapped:

```
         COPY   PROGEQU                        CVT EQUATES
         COPY   TCBEQU                         TCB EQUATES
BIT0     EQU    1                              LEFTMOST BIT
SHIFTCT  DATA   F'0'                           ADDRESS KEY
MAP      DATA   F'0'                           MAPPING WORD
          •
          •
         MOVE   #1,$CFPARM,FKEY=0              GET MAPPING WORD ADDR
         IF     (#1,EQ,0),GOTO,QUIT            NO CSXSYS
         MOVE   MAP,(0,#1),FKEY=0              GET MAPPING WORD
         TCBGET SHIFTCT,$TCBADS                GET ADDRESS KEY
         SHIFTL MAP,SHIFTCT                    SHIFT PARTITION BIT TO 0
         IF     (MAP,ON,+BIT0),GOTO,QUIT       PARTITION NOT MAPPED
          •
          •
QUIT     EQU    *                              TERMINATE PROGRAM
          •
          •
```

Terminating Your Program

Your program is required to terminate when a CP P (stop) or CP H (halt) command designating your program is issued. If your program is a transaction-processing program managed by the program dispatcher, it is also expected to stop when the program dispatcher tells it to. The reasons the program dispatcher tells transaction-processing programs to stop are explained in "Creating a Transaction" on page 67.

Whatever its source, a request to stop or halt is sent to your program as a status message. You receive a status message, as you do any other message, with a RECEIVE M or RECEIVE T instruction. A return code of +6 indicates that the message is a status message.

Your program will receive no status message other than a stop request unless it communicates with circuit stations. In that case, you'll need to distinguish a status message that tells your program to stop from one that contains X.25 control information by its content. A stop request is a 2-byte message—either P or H, followed by a blank.

By the time you receive a status message that is the result of a CP P or CP H command, your program's station block has been flagged inactive and its disk queue, if any, has been closed. As a result, no more data or transaction messages will be placed on your station's message queue, and you can't receive any messages that are queued on disk.

In your termination routine, you may process any messages still queued in storage. Then you must delete your station block and terminate execution. You can do this by issuing the LOCATE ST instruction with OPTION=PROGSTOP. If you issue an EDL PROGSTOP instruction, be sure to specify LOGMSG=NO.

Using Task Error Exits

It is recommended that you provide a task error exit routine that gains control if your program fails. The purpose of the task error exit is to log error information and terminate the failing program with minimal impact on the rest of the system. You can use $$EDXIT, the task error exit routine supplied with EDX, which is described in the *EDL Programming Guide*. You can also code your own task error exit routine, as explained in the *EDX Customization Guide*.

If you code your own task error exit routine, you can use a special form of the SEND E instruction to log the hardware status information on the Communications Facility system log by:

```
SEND E,82[,taskname],ID=C'CF',
TYPE=X,XCODE=teehsa*
```

The message number must be 82; the message ID must be CF; and the message type must be X. Operand XCODE must be an indirect reference to the last word of your task error exit control block, the word that contains the address of your hardware status area.

You don't need to specify *taskname* for a main task (one defined by a PROGRAM statement). If the exit routine is for a subtask defined by a TASK statement, specify its name. If the exit routine is for a subtask started by an ACTIVATE T instruction, specify the associated station name as the task name.

When the SEND E instruction is executed, this information is logged to the Communications Facility system log:

```
*hh:mm:ss CF81 E plp[2] program PROGRAM OR MACHINE CHECK

PSW  IAR  AKR  LSR  R0   R1   R2   R3   R4   R5   R6   R7
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx

 hh:mm:ss CF82 I 0000 program [task] EXECUTION TERMINATED
```

Using Diagnostic Dumps

During testing of a program, you may want to print the contents of some area of storage to see if the program is running correctly. To log diagnostic dumps on the Communications Facility system log, use a SEND ERROR instruction with TYPE=D, so:

```
SEND ERROR,dump-length,'dump-header',XCODE=dump-loc,TYPE=D
```

[2] program load point

The dump-length is the number of 16-byte units (1 to 255) to be dumped. The dump-header is whatever text you want to identify the dump data. XCODE is the address of the area to be dumped.

Assuming DUMPLOC contains 2CA4, this request:

```
SEND ERROR,3,'STORAGE DUMP',XCODE=DUMPLOC*,TYPE=D
```

will produce this result on the system log:

```
hh:mm:ss CF84 D plp program STORAGE DUMP
                2CA4 xxxx xxxx xxxx xxxx...
                2CB4 xxxx xxxx xxxx xxxx...
                2CC4 xxxx xxxx xxxx xxxx...
```

Using Station Blocks

Figure 6 on page 31 shows the format of a user or message station block, the types you're most likely to use with your application programs.

The labeled fields are used by the Communications Facility for the following information:

Q$NIQ The address of the next station block in S$POOL.

Q$PIQ The address of the previous station block in S$POOL.

Q$TYPE The station type; X'02' for a user station; X'0C' for a message station.

Q$STYPE The station subtype; X'00'.

Q$STAT The station's status (16 bits):

```
0123 4567 89AB CDEF
X... .... .... ....  1 = Station active
.XXX XXXX .... ....  Not used
.... .... X... ....  1 = Waiting for acknowledgment of SEND
.... .... .XXX ....  Not used
.... .... .... X...  1 = Input from station prohibited
.... .... .... .X..  1 = Output to station held
.... .... .... ..XX  Not used
```

Q$DLV The network address of the station's direct link.

Q$NAU The station's network address.

Q$TCF The text count field for the station name and disk queue flags.

Q$NAME The station name.

Q$FIQ The address of the first storage-queued message for the station.

Q$LIQ The address of the last storage-queued message for the station.

Q$DQA The address of the file control block, if the station has a disk queue.

Q$ECB An EDX event control block, posted when a message is sent to the station.

```
DEC   HEX
            ----------------------------------------
  0     0   |      Q$NIQ         |      Q$PIQ        |
            ----------------------------------------
  4     4   | Q$TYPE | Q$STYPE   |      Q$STAT       |
            ----------------------------------------
  8     8   |      Q$DLV         |      Q$NAU        |
            ----------------------------------------
 12     C   |      Q$TCF         |      Q$NAME       |
            --------------------                   -
 16    10   |                                      |
            -                   --------------------
 20    14   |                    |                   |
            ----------------------------------------
 24    18   |                    |                   |
            ----------------------------------------
 28    1C   |                    |                   |
            ----------------------------------------
 32    20   |                    |                   |
            ----------------------------------------
 36    24   |      Q$FIQ         |      Q$LIQ        |
            ----------------------------------------
 40    28   |      Q$DQA         |                   |
            ----------------------------------------
 44    2C   |                Q$ECB                   |
            -                   --------------------
 48    30   |                    |      Q$WORK       |
            ----------------------------------------
 52    34   |      Q$ALV         |      Q$MSRC       |
            ----------------------------------------
 56    38   |                Q$ISN                   |
            ----------------------------------------
 60    3C   |                Q$OSN                   |
            ----------------------------------------
 64    40   |                Q$ICNT                  |
            ----------------------------------------
 68    44   |                Q$OCNT                  |
            ----------------------------------------
 72    48   |               Q$ACKECB                 |
            -                   --------------------
 76    4C   |                    |                   |
            ----------------------------------------
```

Figure 6. User or Message Station Block

Q$WORK	The address of the station's work area, if you used the LOCATE ST instruction to create the station and specified a work area.
Q$ALV	The network address of the station's alternate link.
Q$MSRC	A message address or disk flag used by RECEIVE instructions.
Q$ISN	The number of messages sent with this station as the origin.
Q$OSN	The number of messages received from the station's queue.
Q$ICNT	The number of characters sent with this station as the origin.
Q$OCNT	The number of characters received from the station's queue.

Q$ACKECB	An EDX event control block used to acknowledge receipt of messages sent with this station as the origin.

You can use the rest of a user or message station block, including the unused bits in Q$STAT, however you want.

If you need to know the format of other types of station blocks, see the *Debugging Guide.*

Using Message Stations

You can create a message station and use its message queue for any purpose you choose. You might, for example, create a message station if you have a program (program A) creating messages and storing them on disk to be processed later by another program (program B).

The requirements for such a setup are:

- The message station must be defined in $.SYSNET as having a disk queue.
- Program A issues a CP S command to start the message station, sends priority 127 (disk-queued) messages to it, issues a CP P command to stop the message station, and deletes the message station's station block.
- Program B issues a CP S command to start the message station, receives the messages program A put on the queue, issues a CP P command to stop the message station, and deletes the message station's station block.

It would also be possible to have programs A and B running simultaneously. That setup would impose these additional requirements:

- Program A would have to have some means of knowing whether program B is running so it would not stop the station if program B was running. You might, for example, have program B set a bit in the station status word when it begins execution.
- Program B needs to know when to stop. You could handle this requirement by having program A send a special-format "last" message.

If you use message stations, you should be aware that they differ from other types of stations in these ways:

- Messages can be sent to a stopped message station. You have to delete a message station to prevent messages from being sent to it.
- You can't wait for acknowledgment that a message sent to a message station has been received. The wait is satisfied as soon as the message has been put on the station's queue.
- When you use a LOCATE ST instruction to create a message station (as opposed to defining it in $.SYSNET and starting it), the station is treated as a local station, even though the node assignment in its network address is that of a remote node.

Using the Alternate Link

The program dispatcher uses a station's alternate link to handle transactions that it is unable to route. The X.25 IOCP uses the alternate link to send X.25 control messages. If your program communicates with stations whose alternate link is not used for one of these purposes, you can use the alternate link however you want.

Assume, for example, that your program communicates with operators at 3270 terminals or at Series/1 terminals managed as 3277s. The terminal stations are linked to your program, so it receives all input from the operators. Your program processes some of the input and sends some of it on to a host program. The alternate link of each terminal station could be a station that provides a connection to the host program—an emulated 3277 terminal, an emulated channel attach port, or an SNA logical unit.

If the instruction that receives a message from a terminal is:

```
RECEIVE M,BUFFER,ORIGIN=USERTERM
```

then the instructions to send a message on to the host program are:

```
LOCATE ST,#1,USERTERM
LOCATE NA,#2,(Q$ALV,#1),EXIT=NOLINK
SEND M,(Q$NAME,#2),BUFFER,ORIGIN=USERTERM
```

The RECEIVE instruction places the name of the originating station in the USERTERM field. The first LOCATE instruction finds the station block with that name and returns its address in register 1. Field Q$ALV is the alternate link vector—the network address of the alternate link. The second LOCATE instruction finds the station block with that network address and returns its address in register 2. If it doesn't find the station block, control passes to the instruction labeled NOLINK. The destination in the SEND instruction is the name from the alternate link's station block.

Communicating with 3270-Type Terminals

When you write a program that communicates with 3270-type stations, you must understand the data format requirements of the I/O control programs that manage the stations. This section describes those requirements for these types of stations:

- 3270 terminals attached to the Series/1 by a BSC line.
- 3101, 4978, 4980, 7485, and printer devices managed as if they were 3270 terminals.
- Stations used to communicate with host systems over a BSC line (3270 emulation terminals), a channel attachment (port terminals), or an SNA connection (SNA logical units, types 2 and 3). The term "emulated terminal station," as used in this section, means any of these types of stations.

  Note that the term does not include SNA type 1 logical units. These stations receive SNA character string data, not 3270 data streams, from the host.

The I/O control program that manages 3270 terminals can also be used for communication between Series/1s. Communication is from 3270 control terminal stations in one Series/1 to 3270 emulation terminal stations in another Series/1.

BSC Control Characters

The I/O control programs expect a message sent to a 3270-type station to begin with the BSC control character STX (X‘02’, start text) or DLE/STX (X‘1002’, start transparent text). If it doesn’t, the I/O control program appends control information to the beginning of the message, as described under “Unformatted Messages” on page 35.

The I/O control programs check that a message sent to a 3270-type station ends with an ETX (X‘03’, end text) or an ETB (X‘26’, end transmission block). If the last character of the message is neither of these, the I/O control program appends an ETX; if the last character is an ETB, the I/O control program replaces it with an ETX.

The SNA and channel attach I/O control programs remove BSC information from a message before they send it to the host. They append BSC information to data received from the host. This means that messages you exchange with an emulated terminal station have the same data format, regardless of the station type.

3270 Data Streams

When you communicate with real terminals (including 3270 control terminal stations used for communication between Series/1s), you send output data streams to the terminals and receive input data streams from them. When you communicate with a host system through an emulated terminal station, the direction is reversed because the Series/1 appears to the host as if it were a 3270 system. You send input data streams to emulated terminal stations and receive output data streams from them.

An output data stream, including BSC control characters, looks like this:

STX	ESC	*cmd*	*wcc*	orders and data	ETX

ESC
  is the BSC escape control character (X‘27’).

cmd
  is a 3270 command.

wcc
  is a 3270 write control character.

An input data stream, including BSC control characters, looks like this:

STX	*cuda*	AID	*cursor*	orders and data	ETX

cuda
is the terminal's control-unit/device-address. When you create an input data stream to be sent to an emulated terminal station, put two blanks here; the I/O control program fills in the actual *cuda* for the terminal.

AID
is the 3270 attention identifier.

cursor
is the 2-byte cursor address.

Not all 3270 data streams are exactly as shown in these diagrams. An output data stream doesn't always have a write control character or data; it depends on the particular 3270 command. An input data stream doesn't always include the cursor address and data. For detailed information about 3270 data streams, see the *3270 Description and Programmer's Guide.*

Unformatted Messages

When you send a message that does not begin with STX or DLE/STX, the I/O control program assumes that the message is not a 3270 data stream and appends BSC and 3270 control information to the message. Therefore, you must either provide a complete BSC/3270 data stream or omit all the leading control information. The exact data appended by each I/O control program is shown in the *Design and Installation Guide.*

Transparent-Text Mode

Transparent-text mode is used to send messages that contain binary data (any value from X'00' to X'FF'), rather than character data. A transparent message may, for example, be a storage dump or an object program.

A transparent message must begin with DLE/STX (start transparent text). You should end the message with an ETX, rather then letting the I/O control program append one, because the I/O control program always deletes an ETX from the end of a transparent message before sending it to the terminal or host. If the last *data* byte of the message should happen to be X'03', the I/O control program will assume it is an ETX and delete it. If the last data byte is X'26', the I/O control program will assume it is an ETB, replace it with an ETX, and then delete it. If you provide the ETX, the message is sent correctly, no matter what the last data byte is.

There are some device dependencies for transparent messages. You can't send transparent messages to a 3270 terminal attached to a 3271 control unit; the control unit ignores transparent messages. You may or may not be able to send transparent messages to a host system over a BSC line; check with the host system programmer to find out whether transparency is supported. You can send transparent messages to a host system through a channel attachment or SNA connection.

The 3270 control and emulation I/O control programs both accept transparent data from the BSC line, but delete the initial DLE (X'10') before sending the message on. Don't send transparent messages between Series/1s that are connected by 3270 control and emulation unless the messages are transactions routed by the program dispatcher; it ensures transparency of messages. You can always send transparent messages between Series/1s connected by a Local Communications Controller, an HDLC line, or a Series/1-to-Series/1 BSC line.

Host Considerations

Messages sent to an emulated terminal station by a host system must be 3270 data streams, just as if they were being sent to a real 3270 system. At a minimum, the message data must be preceded by a 3270 command sequence; for example, a 3270 write command and write control character. Not all 3270 commands are accepted by I/O control programs. The *Design and Installation Guide* shows the restrictions for each I/O control program.

If the message is sent over a BSC line, it must begin with STX/ESC (start text, escape) and end with an ETX.

Host systems can send transparent messages (binary data) to a Series/1. Transparent mode from the host is controlled by host access methods or programming techniques; consult the documentation of the host system you're using. Note that after a transparent message has been received at the Series/1, it can no longer be identified as transparent. If it was received from a BSC line, the I/O control program drops the initial DLE (X'10', data link escape) before sending the message on. If it was received from a channel attachment or SNA connection, it never had a DLE. The program that receives the message must know that it is receiving binary data.

SNA Considerations

If you write programs that communicate with an SNA host system, you must understand what an SNA session is and the bracket protocol on sessions. To begin communication with the host, a session must be established. You can do this by specifying a logon ID when you define SNA logical unit stations to the Communications Facility; the I/O control program establishes the session when the logical unit is started.

If you don't specify a logon ID, you will receive the SNA logon prompt screen when you start communication with an SNA logical unit, and you must create a 3270 data stream with the appropriate fields to reply to it. The *Design and Installation Guide* shows the format of the SNA logon prompt screen.

SNA LU2 half-duplex protocol prohibits you from sending data to the host until the host program has sent an indication that you may. You receive this indication as a 3270 data stream in which the write control character has the keyboard restore bit (bit 6) on. If you send a message while the host application has the right to send, the message is discarded.

One way to control the session protocol is to use the attention feature to signal the host application that you want to send. To do this, you send a 3270 test-request data stream to your LU after the I/O control program is started. The test-request data stream has the following format:

X'016C61', followed by ETX or STX-data-ETX

A second way to control the session isolates your program from half-duplex protocol. Use the CP F command to set the I/O control program station into non-remove mode. (All LUs will operate in non-remove mode, which may affect the operation of terminals using SNA.)

In non-remove mode, messages received while the host has the right to send are not discarded; they are queued on the Series/1, the host is signaled that the Series/1 wishes to send, and the messages are sent when the Series/1 again has the right to send. You must use the CP F command each time you start the I/O control program station, because it resets the mode to remove—the normal SNA LU2 mode—each time it starts.

When you communicate in non-remove mode, both your program and the host program must be aware of it, because data sent to the host may not be a response to the last message sent to the Series/1.

Host Subsystem Considerations

The subsystem (such as CICS) that is used to implement a host application may use special protocol. For example, the subsystem may send your program 3270 screen images. You must include code to handle them and to send the required response. Consult your host subsystem programmer for guidance.

Intercepting Messages from an I/O Control Program

If you want, you can set up an application program that intercepts all the messages that an I/O control program sends into the system from the stations it controls. You may want to do this, for example, to provide an audit trail of messages by writing the messages to a disk data set before sending them on to their destination.

You can intercept messages from any I/O control program except $.IO0A10. To intercept messages, you must link the I/O control program to your program's station:

CP LINK $.IO*xxxx* *userprog*

Because of the link, the I/O control program sends each message to your program instead of to the station to which the origin station is linked.

You must handle the messages you receive from the Local Communications Controller IOCP ($.IO0AB0) and the X.25 IOCP ($.IO0AB8) differently from those you receive from other IOCPs.

Intercepting Messages from $.IO0AB0

When you receive a message from $.IO0AB0, the first 20 bytes are bytes 4-23 of the message header (all of the header except fields M$NIQ and M$PIQ); the message data begins at the 21st byte.

After processing the message, your program might send it on to its destination. To do so, you have to move the message header to a 24-byte area, modify the buffer header to address the message data, and use the HEADER operand of the SEND instruction. For example:

```
BUFF    DEFINE    BUFFER,SIZE=2048,P2=BCOUNT,P3=BADDR
MHDR    DATA      12A(*-*)
MPRI    EQU       MHDR+4
MHDRL   EQU       20
        •
        •
        •
        RECEIVE   M,BUFF
        MOVE      MPRI,BUFF,(+MHDRL,BYTES)
        ADD       BADDR,+MHDRL
        SUB       BCOUNT,+MHDRL
*   PROCESS MESSAGE DATA
        •
        •
        •
        SEND      M,,BUFF,HEADER=MHDR
*   RESET BUFFER DATA ADDRESS
        SUB       BADDR,+MHDRL
```

Intercepting Messages from $.IO0AB8

The first 24 bytes of the messages you receive from $.IO0AB8 are the Communications Facility message header; the message data begins at the 25th byte. You can skip past the message header by modifying the buffer header so that B$ADDR addresses the message data and B$COUNT contains the message length. You must do this if you want to send a message on to its destination after you have processed it. After modifying the buffer header, issue a SEND M instruction with a HEADER= operand that addresses the message header at the start of the buffer. This causes the message to be dispatched to the destination station specified in the message header.

You may receive three kinds of messages from $.IO0AB8. Their type and format vary depending on the usage attribute of the origin circuit station:

- Data messages from circuit stations with usage CF.
- Data messages that begin with an X.25 header from circuit stations with usage STD or STD+.
- X.25 control messages that begin with an X.25 header from circuit stations with usage STD+.

$.IO0AB8 sends both forms of data messages as Communications Facility data messages; your receive instruction completes with return code -1. $.IO0AB8 sends the X.25 control messages as Communications Facility status messages; your receive instruction completes with return code +6.

If you have both CF and STD or STD+ circuit stations, you may want to know whether a data message begins with an X.25 header. If so, use the origin network address in the message header to locate the origin station and determine its usage. The first byte of an X.25 header contains the length of the header. You can use the length to skip past the X.25 header to the message data.

You can skip past the X.25 header in control messages too. You probably won't want to do this, because the header includes the control message type. If you need information about X.25 control messages, see "Writing an X.25 Application Program" on page 107.

The following example shows how to receive all three forms of messages from $.IO0AB8, distinguish between them, and send them on to their destination.

```
BUFF      DEFINE  BUFFER,SIZE=256               INPUT BUFFER
USAGE     DATA    F'0'                          CIRCUIT USAGE
HDRLEN    DATA    F'0'                          X.25 HEADER LENGTH
LOCALHDR  DATA    12F'0'                        LOCAL MSG HEADER

STOP      DATA    C'P '                         CP P ISSUED FOR PROGRAM
HALT      DATA    C'H '                         CP H ISSUED FOR PROGRAM
RCODE     DATA    F'0'                          RETURN CODE
RCDATA    EQU     -1                            DATA MSG RETURN CODE
RCSTAT    EQU     +6                            STATUS MSG RETURN CODE
          •
COPY     S$CFEQU
          •
          RECEIVE M,BUFF, HEADER=LOCALHDR       RECEIVE MESSAGE
          TCBGET  RCODE,$TCBCO                  GET RETURN CODE
          IF      (RCODE,EQ,+RCDATA)            IF DATA MESSAGE
            LOCATE  NA,#1,LOCALHDR+M$OAF,EXIT=NOORG  FIND ORIGIN STATION
            MOVE    USAGE,(Q$USE,#1)            SAVE CIRCUIT USAGE
            ADD     BUFF+B$ADDR,+M$HDRLN        SKIP PAST MSG HEADER
            SUB     BUFF+B$COUNT,+M$HDRLN       ADJUST LENGTH TO MATCH
            MOVE    #1,BUFF+B$ADDR              #1 POINTS TO DATA MSG
            MOVE    #2,BUFF+B$COUNT             #2 IS ITS LENGTH
            IF      (USAGE,NE,+Q#UCF)           IF USAGE IS STD/STD+
              MOVE    HDRLEN+1,(0,#1),BYTE      GET X.25 HDR LENGTH
              ADD     #1,HDRLEN                 SKIP PAST X.25 HEADER
              SUB     #2,HDRLEN                 ADJUST MESSAGE LENGTH
            ENDIF
            •
            •                               PROCESS DATA MESSAGE
            •
          ELSE
            IF        (RCODE,EQ,+RCSTAT)        IF STATUS MESSAGE
              IF        (BUFF,EQ,STOP,1),OR,(BUFF,EQ,HALT,1),GOTO,END
              ADD     BUFF+B$ADDR,+M$HDRLN      SKIP PAST MSG HEADER
              SUB     BUFF+B$COUNT,+M$HDRLN     ADJUST LENGTH TO MATCH
              MOVE    #1,BUFF+B$ADDR            #1 POINTS TO CTL MSG
              MOVE    #2,BUFF+B$COUNT           #2 IS ITS LENGTH
              •
              •                             PROCESS CONTROL MESSAGE
              •
            ELSE
              •
              •                             PROCESS RECEIVE ERROR
              •
            ENDIF
          ENDIF
          SEND    M,,BUFF,HEADER=BUFF           SEND MESSAGE ON
          SUB     BUFF+B$ADDR,+M$HDRLN          RESET BUFFER DATA ADDR
```

You use a SEND M instruction to send all messages on. A bit in the message header identifies it as a Communications Facility data or status message, so the destination will get the appropriate return code when it receives the message.

You may, if you wish, have one program that intercepts data messages from $.IO0AB8 and another that intercepts control messages. To do so, define the data message intercept program as $.IO0AB8's direct link and the control message intercept program as its alternate link. If $.IO0AB8 has a direct link but no alternate link, it sends all messages—data and control—to its direct link.

Intercepting Messages from Other I/O Control Programs

When you receive a message from an IOCP other than $.IO0AB0 or $.IO0AB8, you receive just the message data. Use the ORIGIN operand of the RECEIVE instruction to get the name of the station that originated the message:

```
RECEIVE M,BUFFER,ORIGIN=USERTERM
```

Having processed the message, your program might send it on to the station to which the origin is linked:

```
SEND M,,BUFFER,ORIGIN=USERTERM
```

return it to the origin:

```
SEND M,USERTERM,BUFFER
```

or send it to some other station:

```
SEND M,'ELSEWHER',BUFFER,ORIGIN=USERTERM
```

Providing a Central System Log

Each node in a Communications Facility configuration can have a system log at which all error and informational messages issued at that node are recorded. If your configuration consists of more than one node, you may want to have all messages logged at a central site.

There is a simple way to set up a central log when the nodes are connected by a Local Communications Controller, an HDLC line, or a Series/1-to-Series/1 BSC line. You can assign a printer at the central site to be the log device. At each node, you would define that printer as a device station and set that station as the system log.

There are, however, two disadvantages to this simple solution:

- Messages are logged only at the central site, not at the node where they were issued.
- The log does not show which node a message came from.

An effective solution requires that you write two programs—one to run at the central node and the other to run at each remote node. At each node, define the appropriate program as a user station and add a start command for that station to the $.SYSIPL data set.

Each program should begin by issuing a SEND CP instruction to set itself as the system log:

SEND CP,,'SET LOG *program-name*'

The programs will receive formatted log messages, up to 80 bytes long. The remote node program can use a PRINTEXT instruction to log the message locally, if you want. It should then prefix the message with a unique node identifier and send it to the program at the central node.

The central node program will receive these messages and its own node's log messages. It should prefix its own messages with its own node identifier and use a PRINTEXT instruction to log all messages, with their node identifier, on the log device.

Each program must terminate when it receives a status message, as described under "Terminating Your Program" on page 28. Before doing so, it should issue a SEND CP instruction to turn off logging. Alternatively, the central node program could assign the log device as the system log.

It is strongly recommended that all stations that receive log messages have disk queues. Otherwise a communication line error, for example, can quickly flood the system message pool with error messages. Such a situation could cause a deadlock when, for example, a remote node log program can't send a message to the central node program because the system message pool is full of unreceived messages.

The disk queues should be large to avoid losing log messages during periods of high activity.

When the log messages are sent across nodes, the station to which messages for the remote node are queued should also have a disk queue. The relevant station for different types of connections is:

- Local Communications Controller: The node station that represents the remote node.
- Series/1-to-Series/1 BSC line: The line station (subtype CPU) to which the station that represents the remote node is linked.
- HDLC line: The circuit station (with usage CF) to which the station that represents the remote node is linked.
- Multipoint BSC line: A 3270 emulation terminal station. Note that for this type of connection, the remote node log program would send messages (or be direct linked) to a 3270 emulation terminal station; in the central node, the corresponding 3270 control terminal stations would be direct linked to the central node log program.

Processing Undeliverable Messages

Instead of using utility program $.UT2 to process undeliverable messages, you can write a program that receives undeliverable messages and manages the undeliverable message station, $.WASTE. For example, you might want to notify the operator of undeliverable messages as they occur, so the operator can correct the problems.

Each undeliverable message results in a pair of messages being sent to $.WASTE. The first message of a pair contains the date, the time, and a reason code that indicates why the message could not be delivered. Its format is:

Position	**Content**
1-8	The date, in the form mm/dd/yy
9	A blank
10-17	The time, in the form hh:mm:ss
18	A blank
19-22	The reason code, as documented for utility program $.UT2 in the *Operator's Guide.*

The second message of each pair is the undeliverable message. You will probably want to specify the HEADER operand on the instruction that receives this second message so that you can determine the origin and destination. (You can also receive the headers of the reason messages, but they don't contain any useful information.)

The Communications Facility control program operates as if there is no program managing station $.WASTE. When the Communications Facility is shut down (by a stop or halt command for station $.CF or $.DISP), it issues a stop command for station $.WASTE and then purges the station block. These actions close $.WASTE's disk queue if it has one, or purge any storage-queued messages if it doesn't. As a result, your program doesn't have a chance to process pending undeliverable messages. If this is not acceptable, you can manage shutdown differently by coordinating your operational procedures with the design of your program.

One solution is to have the operator wait until there are no pending undeliverable messages before shutting down the Communications Facility. First, use the CP Q * command to determine whether messages are pending. The FIQ field is 0000 when there are no storage-queued messages; the DISK MSG field is NO when there are no disk-queued messages. If you use this approach, define $.WASTE as a user station and name your program $.WASTE. Give the program a high priority so that it will receive the status message that results from the stop command before its station block is purged. When the program receives the status message, it need only terminate execution.

An alternative solution, when $.WASTE has no disk queue, is to have the operator stop $.WASTE before shutting down the Communications Facility. Define $.WASTE as a user station and name your program $.WASTE. When the program receives the status message that tells it to stop, it can process pending storage-queued messages, issue a log message to notify the operator that processing of undeliverable messages is completed, purge the $.WASTE station block, and terminate execution.

The last solution won't work when $.WASTE has a disk queue, because the stop command closes the disk queue. In this case, define $.WASTE as a message station, give your program some other name (for example, WASTEPGM), and define it as a user station. Have the operator stop WASTEPGM instead of $.WASTE before shutting down the Communications Facility. The program has to receive a status message from its own queue and undeliverable messages from the $.WASTE queue.

You might write the program as two tasks. The main task should attach a subtask and then wait to receive the status message that tells it to stop. When it does, the main task should wait for the subtask to finish and then issue a log message, purge the $.WASTE and WASTEPGM station blocks, and terminate execution. The subtask should receive and process messages from $.WASTE's queue until there are no more messages, and station WASTEPGM has been stopped (bit 0 of Q$STAT in the station block is 0).

Creating and Managing 3270 Panels

The Communications Facility includes an interactive utility program, $.PANEL, which you can use to design panels that are to be displayed as part of Communications Facility applications. A panel is a screen image for a Communications Facility terminal, either a 3270 display station or a 3101, 4978, 4980, or 7485 terminal being managed as if it were a 3277. The Communications Facility also includes a subroutine, S$GETPNL, which you can use in your program to fetch the panels you created through $.PANEL. You can use the language extension GET and PUT instructions with the panel fetch subroutine to append variable data to panels and to retrieve the input entered by the user of your program.

You can get a printed description of a panel when you create it. You can also use the utility program $.PNLUT1 to print descriptions of panels.

Using $.PANEL to Design Panels

$.PANEL is itself a Communications Facility application program that can communicate with multiple users. The maximum number of concurrent users depends on the amount of dynamic storage defined in the PROGRAM statement. The program is distributed with 512 bytes of dynamic storage, which allows four concurrent users (each active user requires 116 bytes). You can use the SS command of the EDX utility $DISKUT2 to change the amount of dynamic storage.

You must define $.PANEL to the Communications Facility as a user station, with station name $.PANEL.

Terminal Considerations

You can use $.PANEL at any Communications Facility terminal: 3101, 4978, 4980, 7485, or 3270 display station. The terminal may be in the same node as $.PANEL or in a remote node. The target terminal (the one at which the panel is displayed) may be of the same or a different type. When creating any panel, you can specify any 3270 function supported by $.PANEL; but you should keep in mind that the type of target terminal may affect what is actually displayed when the panel is used. For example, you can define the intensity of a field as normal, bright, or nondisplay. On the 7485, the field is displayed in the intensity you have defined. Since a 3101 terminal has only one level of intensity, normal and bright fields appear the same. On a 4978, unprotected fields (those in which the application user can enter data) display at bright intensity and protected fields (those in which the application user can't enter data) display at normal intensity; therefore, the protected/unprotected attribute of a field overrides its normal/bright attribute.

The *Design and Installation Guide* describes the 3270 functions that are emulated by the 3101, 4978, 4980, and the 7485 I/O control programs. If you need more information about 3270 concepts and functions, see the *3270 Description and Programmer's Guide.*

$.PANEL Output

The output of $.PANEL is a member of a partitioned data set. The data sets used for $.PANEL output must be allocated and initialized using the EDX utilities $DISKUT1 and $DIUTIL. $.PANEL allocates the members; each member requires nine records. A member contains a 3270 data stream and, optionally, a field table. The 3270 data stream, when sent to a Communications Facility terminal, displays the defined panel. The field table contains, for each field specified by the panel's designer, the field's screen location, length, and type. You can use this information in your application programs to simplify the use of PUT F and GET F instructions when the panel is processed.

Changing Member Size

If you're defining panels with many fields, nine records may not be enough. In that case, you will get an error message, "PANEL EXCEEDS MEMBER SIZE" or "TOO MANY NON-STANDARD FIELDS". You can increase the size of the member that $.PANEL allocates for new panels. This has no effect on the size of existing smaller panels, but you can still use $.PANEL to process them.

Each record you add will allow for 64 more nonstandard fields.

To change the number of records, change the following instructions in the modules S$PANEL and S$PNLPRT:

Module	**Label**	**Operand**
S$PANEL	BUFF1	SIZE=*n* × 256 [3]
S$PANEL	BUFF2	SIZE=*n* × 256
S$PANEL	P#REC	*n*
S$PNLPRT	BUFF	SIZE=*n* × 256
S$PNLPRT	BUFFL	*n* × 256

Assemble the modules and build load modules $.PANEL and $.PNLPRT, as explained in the chapter "Maintaining the Communications Facility" in the *Design and Installation Guide.*

You also need to increase the dynamic storage of the utility program $.PNLUT1. Use the SS command of the EDX utility $DISKUT2 to set the size to the new member size plus 10.

Overview of Panel Design

Using $.PANEL to design a panel is a five-phase process. For each phase, the program presents you with a panel in which you enter the required information. Where appropriate, the program provides default information.

The five phases are:

1. Panel Specification: Identify the panel to be defined.

2. Design Indicators: Define indicators used during the panel layout phase to delimit fields.

3. Panel Layout: Define the contents of the panel.

4. Field Attributes: Define the attributes of the fields that make up the panel.

5. Output Options: Define options that apply to the entire panel.

[3] *n* is the number of records per member to be allocated.

You proceed from phase to phase with the ENTER key and program function (PF) keys.

PF1 causes the display of help information for the current phase.

PF12 causes work on the current panel to end. A working copy of the panel is saved, but it is not in the form that application programs can use. The panel will be saved in its final form only when you complete the last phase (output options) by pressing the ENTER key. During a session with $.PANEL, you can use either form of the panel.

The meaning of the other PF keys and the ENTER key varies with the different phases, and is shown in the lower left part of the screen at all times.

The action defined for ENTER also applies to any PF keys for which no action is defined. For example, look at Figure 7 on page 48, which shows what the screen looks like during the panel specification phase. During this phase, PF2 and PF6-11 cause the program to move on to the design indicators phase.

When you press PF1 or PF12, any pending input is discarded. When you press any other PF key or ENTER, your input is processed before the action defined for the key occurs.

If your input is in error, a message appears at the right side of the bottom row. Explanations of the messages are in the section "$.PANEL User Messages" on page 59.

The panel data that you specify in the panel layout phase may include lowercase alphabetic characters. For all other input, lowercase characters are converted to uppercase.

The following sections explain each phase of the panel design process. The panel used to illustrate the process is CFMENU. This is the panel used by the Communications Facility sample program, $.CFMENU, which is described in the chapter Appendix A, "$.CFMENU Sample Program" on page 197.

Starting a Session with $.PANEL

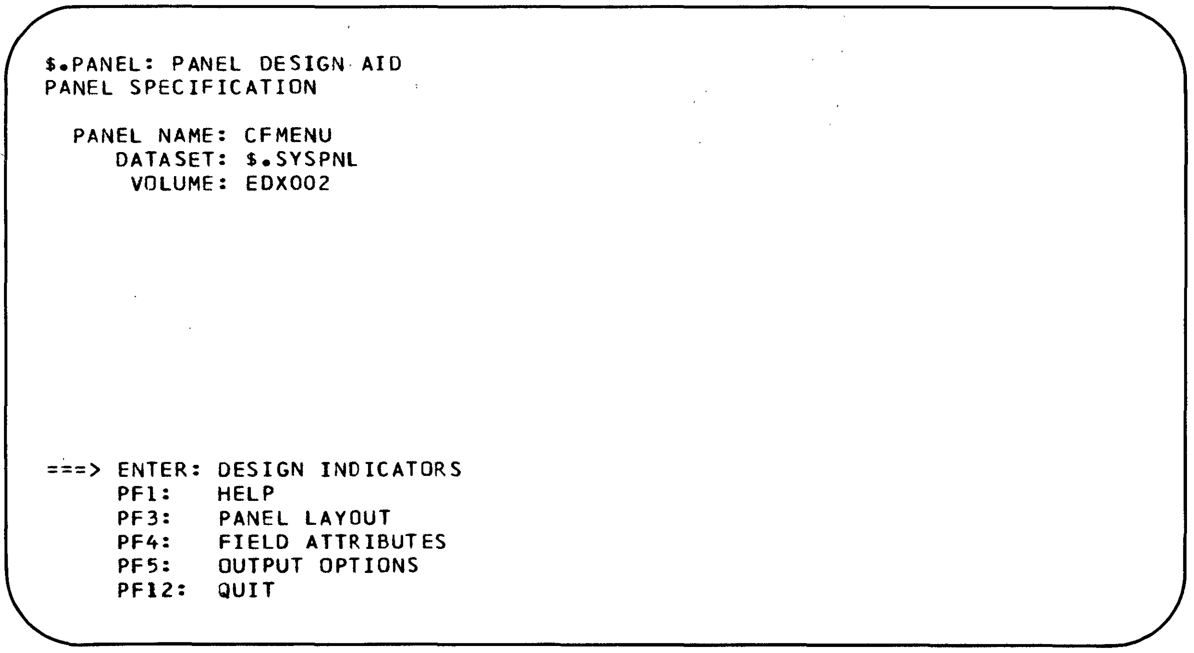
To start a session with $.PANEL, you must start the program, link your terminal to it, and start your terminal. The following commands will accomplish this process:

```
> CP S $.PANEL
> CP LINK staname $.PANEL
> CP S staname
```

Then press ENTER and proceed with the first phase of panel design.

Panel Specification Phase

Figure 7 on page 48 shows the layout of the screen during the panel specification phase. Enter the panel name (1 to 8 characters), the data set name, and the volume name. The data set must have been allocated and initialized. If the panel (that is, the member) does not already exist, $.PANEL allocates it. If the panel does exist, you may modify it during this session.



```
$.PANEL: PANEL DESIGN AID
PANEL SPECIFICATION

  PANEL NAME: CFMENU
     DATASET: $.SYSPNL
      VOLUME: EDX002









===> ENTER: DESIGN INDICATORS
     PF1:   HELP
     PF3:   PANEL LAYOUT
     PF4:   FIELD ATTRIBUTES
     PF5:   OUTPUT OPTIONS
     PF12:  QUIT
```

Figure 7. Screen Layout During Panel Specification

Design Indicators Phase

Design indicators are four characters you will use during the panel layout phase to delimit fields and define their types. Three of them define field types, and the fourth is a fill character.

There are three types of fields; you specify a design indicator for each:

- *Standard:* A protected field (that is, one in which the user of your application is not allowed to enter data) that has default attributes (shown in Figure 12 on page 55). You specify only its contents.

- *Protected:* A field, in which the application user is not allowed to enter data, that has nondefault attributes, which you specify during the field attributes phase.

- *Unprotected:* A field in which the application user can enter data. You may specify nondefault attributes for it.

The *field fill* design indicator is used to indicate trailing field positions that are to appear as blanks in the panel.

Figure 8 on page 50 shows how the screen appears during the design indicators phase. The design indicators shown in the figure (& for standard, ! for protected, @ for unprotected, and / for field fill) are the defaults.

You can change the design indicators to characters other than the defaults.

The following rules apply:

- The four indicators must be unique.
- The three field start indicators cannot be used within fields.
- The field fill indicator can be used within fields; only the rightmost ones in a field appear as blanks in the panel.

Panel Layout Phase

The panel layout phase is the phase during which you define what will actually appear to the panel's user. A set of editing commands, described under "Panel Layout Editing Commands" on page 50, is provided to help you in manipulating the panel.

During this phase, the layout area is marked with column numbers across the top and row numbers down the left, as shown in Figure 9 on page 51. The panel you create will be 80 columns wide and 24 rows long. Because some of your screen is taken up by the row and column numbers and the PF key explanations, you can't see your whole panel at one time. The area you see at one time is a *window* to part of your panel. Use PF3 to move the window from the left side of the panel to the right side, or vice versa. Use the TOP and BOT commands (described under "Panel Layout Editing Commands" on page 50) to move the window up and down.

Enter your panel data in the layout area, using the design indicators to delimit fields. Your four indicators are shown in the center of the bottom row, in the sequence standard, protected, unprotected, and fill.

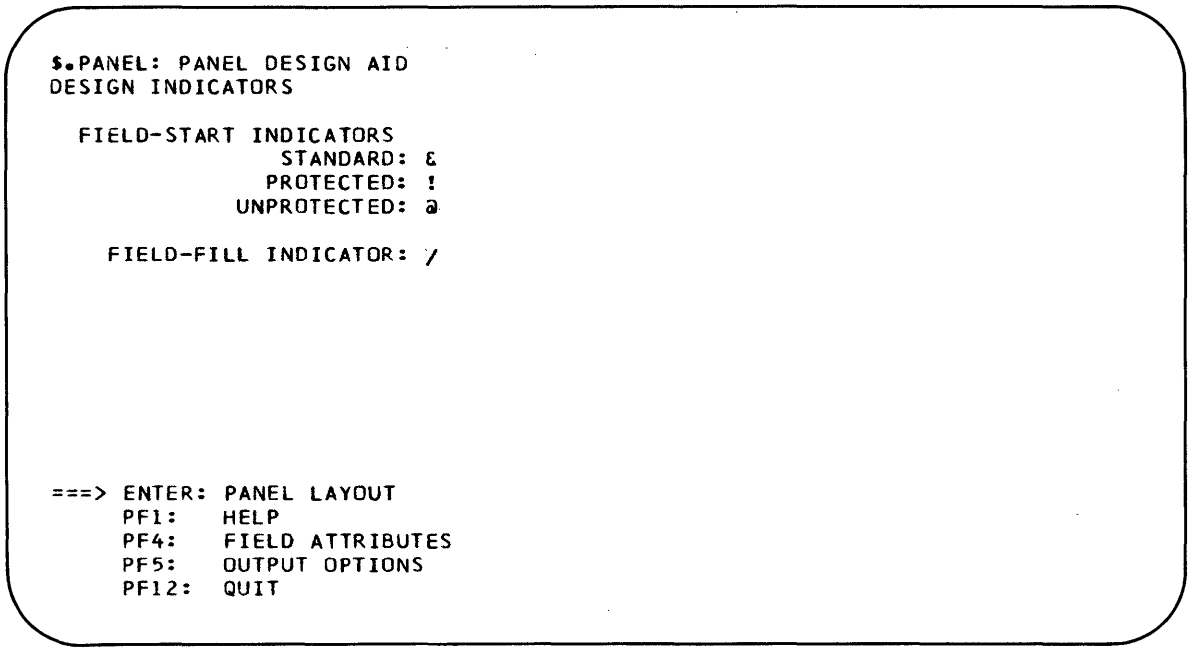
A field extends from the position following a field start indicator to the rightmost nonblank character preceding the next field start indicator (or the end of the panel). A field may wrap from one row to the next, but not from the bottom row to the top row. A field may be from 0 to 254 characters long, not counting the field start indicator. In the resulting panel, the positions occupied by field start indicators and all undefined positions are protected and displayed as blanks.

Some examples of panel data, using the default design indicators, are:

&DEMO MENU	A 9-character protected field.
&(Y/N)	A 5-character protected field; the slash is not a fill character here, because it is not in a trailing position.
@////////	An 8-character blank unprotected field.
@SAMPLE//	An 8-character unprotected field with a 6-character initial value.
!	A null (length 0) protected field that is to have nondefault attributes.

Panel Layout PF and ENTER Keys

When you press ENTER, you remain in the panel layout phase. Data entered in the panel layout area and commands are read and processed.



```
$.PANEL: PANEL DESIGN AID
DESIGN INDICATORS

  FIELD-START INDICATORS
                STANDARD: &
               PROTECTED: !
             UNPROTECTED: @

    FIELD-FILL INDICATOR: /






===> ENTER: PANEL LAYOUT
     PF1:   HELP
     PF4:   FIELD ATTRIBUTES
     PF5:   OUTPUT OPTIONS
     PF12:  QUIT
```

Figure 8. Screen Layout During Design Indicators Phase

Use PF2 to display the panel as the user will see it. You may modify the display to check cursor movement and field attributes, but these changes are not processed. Press ENTER (or any PF key) to leave the display and resume panel design. If you pressed PF2 while working on the panel shown in Figure 9 on page 51 , you would then see the panel shown in Figure 10 on page 52.

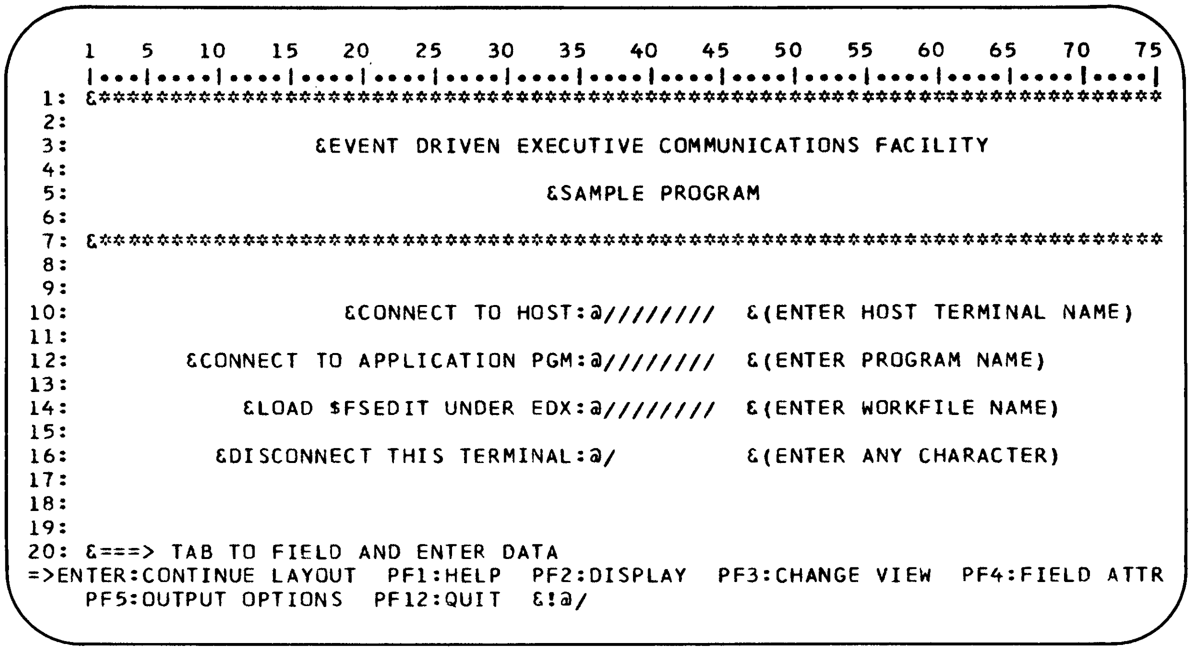
During panel design, it is convenient to alternate between the panel layout and field attributes phases. Use PF4 to transfer to the field attributes phase.

Panel Layout Editing Commands

You enter the panel layout editing commands in the row number fields on the left side of the panel layout screen, including the blank field above the top row number. A command applies to the row in which it is entered; the blank field is, in effect, the top row less one.

The commands are:

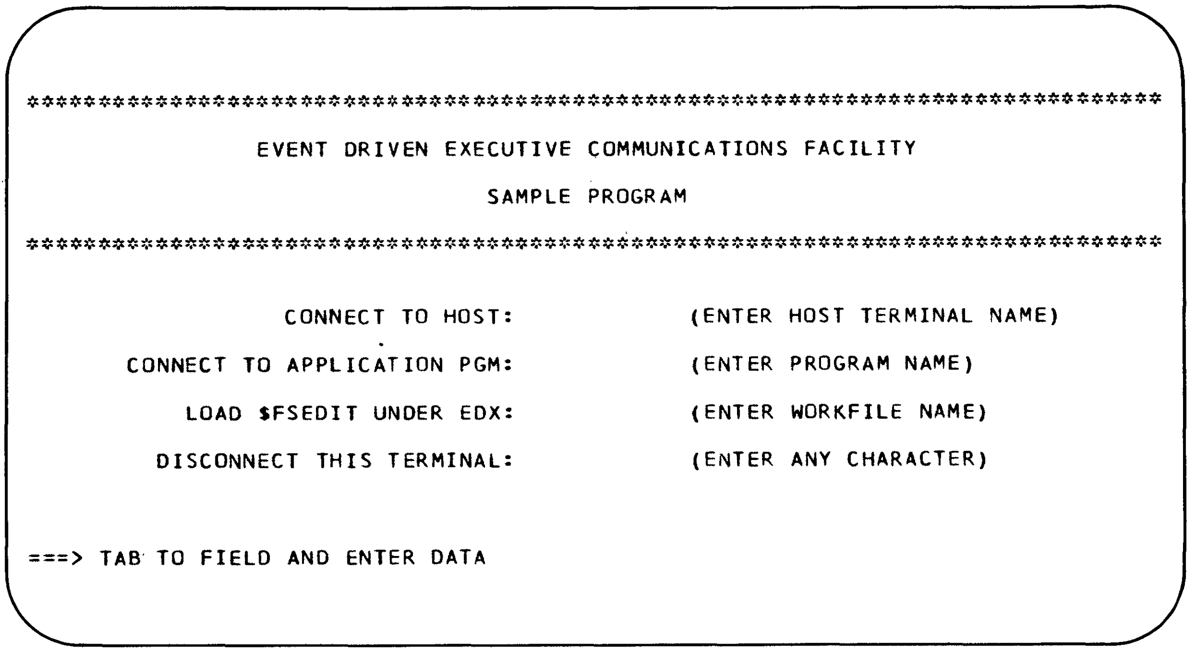
TOP	Scroll this row to the top of the layout area. TOP causes the panel to move up in the window, but the last row of the panel never appears above the last row of the layout area. If, for example, rows 1 to 20 of a 24-row panel are in view, TOP on any row from 5 to 20 causes rows 5 to 24 to appear.
BOT	Scroll this row to the bottom of the layout area. BOT causes the panel to move down in the window, but the first row of the panel never appears below the first row of the layout area.
L*nn*	Shift this row left *nn* columns. This causes the entire row, not just the portion that appears in the window, to be shifted; blanks are inserted on the right. To clear a row, specify L80.



```
    1   5   10   15   20   25   30   35   40   45   50   55   60   65   70   75
    |...|....|....|....|....|....|....|....|....|....|....|....|....|....|....|
 1: &**************************************************************************
 2:
 3:                 &EVENT DRIVEN EXECUTIVE COMMUNICATIONS FACILITY
 4:
 5:                                 &SAMPLE PROGRAM
 6:
 7: &**************************************************************************
 8:
 9:
10:                   &CONNECT TO HOST:@///////// &(ENTER HOST TERMINAL NAME)
11:
12:        &CONNECT TO APPLICATION PGM:@///////// &(ENTER PROGRAM NAME)
13:
14:            &LOAD $FSEDIT UNDER EDX:@///////// &(ENTER WORKFILE NAME)
15:
16:          &DISCONNECT THIS TERMINAL:@/         &(ENTER ANY CHARACTER)
17:
18:
19:
20: &===> TAB TO FIELD AND ENTER DATA
=>ENTER:CONTINUE LAYOUT  PF1:HELP  PF2:DISPLAY   PF3:CHANGE VIEW   PF4:FIELD ATTR
    PF5:OUTPUT OPTIONS  PF12:QUIT  &!@/
```

Figure 9. Screen Layout During Panel Layout

R*nn*	Shift this row right *nn* columns. This causes the entire row to be shifted; blanks are inserted on the left.
CEN	Center this row. This causes the row to be shifted left or right, so that there are equal numbers of leading and trailing blanks.
D	Delete this row.
DD	Delete a block of rows encompassed by a pair of these commands. For example, to delete rows 5 through 10, enter DD on row 5 and on row 10.
I*nn*	Insert *nn* blank rows after this one.
"*nn*	Duplicate this row *nn* times.
M	Move this row.
MM	Move the block of rows encompassed by a pair of these commands.
C	Copy this row.
CC	Copy the block of rows encompassed by a pair of these commands.
A	Move or copy after this row.
B	Move or copy before this row.



```
*****************************************************************************

                 EVENT DRIVEN EXECUTIVE COMMUNICATIONS FACILITY

                                 SAMPLE PROGRAM

*****************************************************************************


                  CONNECT TO HOST:            (ENTER HOST TERMINAL NAME)

         CONNECT TO APPLICATION PGM:            (ENTER PROGRAM NAME)

             LOAD $FSEDIT UNDER EDX:            (ENTER WORKFILE NAME)

           DISCONNECT THIS TERMINAL:            (ENTER ANY CHARACTER)


===> TAB TO FIELD AND ENTER DATA
```

Figure 10. Panel Display During Panel Layout Phase

For commands of the form X*nn* (where X represents a command), *nn* is optional; the default is 1. To specify a 1-digit value, type a blank after it (or type X0*n*). If the row number field contains 'X*n*:', *n* is assumed to be part of the original row number, and the default is taken.

A request that requires more than one command (block commands, move after/before, and copy after/before) must be complete on one screen. You may enter more than one command at a time, except that only one move or copy is allowed. Invalid commands are ignored.

Remember that a panel contains only 24 rows. Insert, duplicate, and copy commands cause rows to be dropped from the bottom of the panel. The delete command causes blank rows to be appended to the bottom of the panel.

When you press ENTER (or any PF key except 1 or 12), input is processed in this sequence:

- Changes to the panel layout area.
- Commands other than move or copy, from top to bottom.
- Move or copy command.

The following examples illustrate the effect of some of the commands. In each example, a portion of the layout screen with commands entered but not yet processed is shown on the left; the result after the commands are processed is shown on the right.

```
20: &AAAAA        20: &AAAAA
I1: &BBBBB        21: &BBBBB
22: &CCCCC        22:
D3: &DDDDD        23: &CCCCC
24: &EEEEE        24:
```

The insert command on row 21, which causes row 24 to be dropped, is processed before the delete command on row 23.

```
"0: &AAAAA        20: &AAAAA
21: &BBBBB        21: &AAAAA
22: &CCCCC        22: &BBBBB
L3  &DDDDD        23: &CCCCC
24: &EEEEE        24: DDD
```

Although commands are processed from top to bottom, they apply to the row on which they are entered. Thus, row 23 is shifted even though the preceding duplicate command causes it to become row 24.

```
MM: &AAAAA        20: &EEEEE
21: &BBBBB        21: &AAAAA
R5: &CCCCC        22: &BBBBB
MM: &DDDDD        23:  &CCCCC
A4: &EEEEE        24: &DDDDD
```

Row 22 is shifted only one column; the 5 is assumed to be part of the original row number. The move command is processed last.

Attributes of fields are retained when they are moved or duplicated as a result of commands. Attributes are not retained when fields are moved as a result of changes to the panel layout area.

If, for example, you want to shift this row:

```
10:         &PRICE:@//////      &QUANTITY:@////
```

so that it looks like this:

```
10:       &PRICE:@//////      &QUANTITY:@////
```

there are three ways to do it: retype the row, use the delete key of the terminal to delete two of the leading blanks, or enter a shift left command.

In the third case, any nondefault attributes that you have specified for the unprotected fields are retained. In the first two cases, they are not; the fields are assumed to be new ones and are given default attributes.

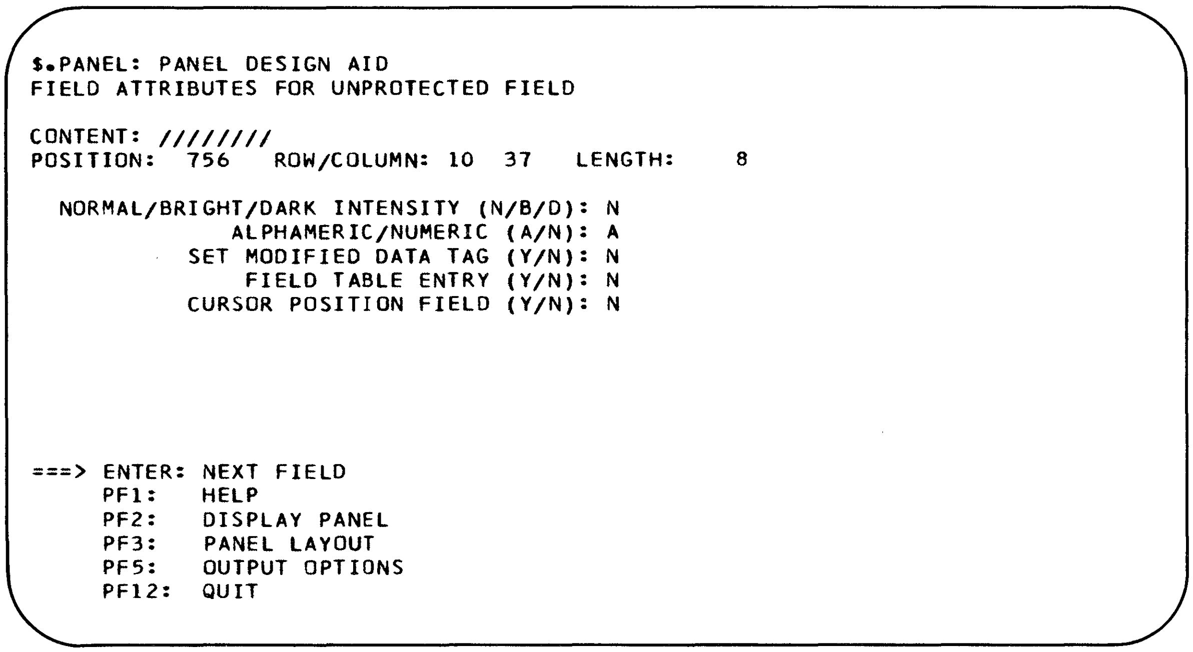
Field Attributes Phase

During the field attributes phase, the program presents you with attributes of each nonstandard (unprotected or protected) field, one field at a time in panel layout sequence. The program does not present fields that you define as standard. Press ENTER to proceed from one field to the next, or from the last field to the first.

The phase begins with the first field except when you enter from the panel layout phase with the cursor at the field start indicator of a nonstandard field; then that field is presented. When you transfer to panel layout from this phase (using PF3), the cursor is positioned at the current field of this phase.

Use PF2 to display the panel as the user will see it, as described under "Panel Layout Phase" on page 49.

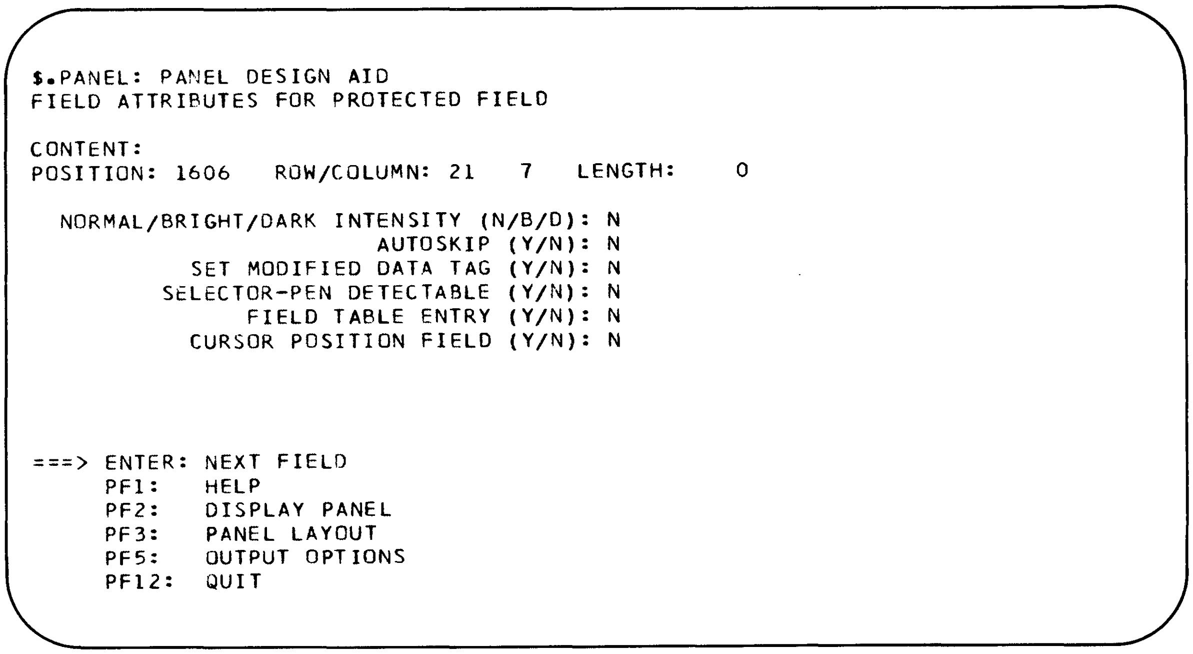
Figure 11 shows the attributes for an unprotected field, the first one of the sample panel. The two rows above the set of attributes (field content, position, and length) are for information only; you can't modify them here. Note that the screen location of the start of the field data is shown in two ways: its position relative to 0, and the corresponding row and column. The attributes shown in Figure 11 are the defaults for an unprotected field. In the completed sample panel, each unprotected field has a field table entry, and the first one is the cursor position field.



```
$.PANEL: PANEL DESIGN AID
FIELD ATTRIBUTES FOR UNPROTECTED FIELD

CONTENT: ////////
POSITION:  756   ROW/COLUMN: 10  37   LENGTH:     8

  NORMAL/BRIGHT/DARK INTENSITY (N/B/D): N
               ALPHAMERIC/NUMERIC (A/N): A
           SET MODIFIED DATA TAG (Y/N): N
                FIELD TABLE ENTRY (Y/N): N
           CURSOR POSITION FIELD (Y/N): N





===> ENTER: NEXT FIELD
     PF1:   HELP
     PF2:   DISPLAY PANEL
     PF3:   PANEL LAYOUT
     PF5:   OUTPUT OPTIONS
     PF12:  QUIT
```

Figure 11. Unprotected Field Attributes

Figure 12 on page 55 shows the attributes for a protected field, the only nonstandard protected field in the sample panel. The attributes shown are the defaults. In the completed sample panel, this field has a field table entry. This is the field in which the sample program writes error messages. Note that it is defined as null, because the sample program provides the field contents. The purpose of defining the field as part of the panel is to establish its field attributes and screen location.



```
$.PANEL: PANEL DESIGN AID
FIELD ATTRIBUTES FOR PROTECTED FIELD

CONTENT:
POSITION: 1606   ROW/COLUMN: 21    7   LENGTH:     0

  NORMAL/BRIGHT/DARK INTENSITY (N/B/D): N
                        AUTOSKIP (Y/N): N
          SET MODIFIED DATA TAG (Y/N): N
        SELECTOR-PEN DETECTABLE (Y/N): N
               FIELD TABLE ENTRY (Y/N): N
           CURSOR POSITION FIELD (Y/N): N




===> ENTER: NEXT FIELD
     PF1:   HELP
     PF2:   DISPLAY PANEL
     PF3:   PANEL LAYOUT
     PF5:   OUTPUT OPTIONS
     PF12:  QUIT
```

Figure 12. Protected Field Attributes

The meaning of the various attributes is explained in the remainder of this section, as they apply to a real 3270 terminal. Remember that the result may be different if the panel is used at a 3101, 4978, 4980, or 4985 terminal, as discussed in section "Using $.PANEL to Design Panels" on page 45.

Intensity:
A field can be of normal, bright, or dark (nondisplay) intensity.

Alphameric/Numeric:
The application user can enter any character in an alphameric field. The user can enter only digits, minus, and decimal point in a numeric field.

Automatic Skip:
This attribute is relevant only for a field that immediately follows an unprotected field. If you specify "Y", when the user enters a character in the last position of the preceding unprotected field, the cursor skips to the next unprotected field. If you specify "N", the cursor skips to the beginning of this (the nonautomatic skip) field.

Modified Data Tag:
If you specify "Y", the contents of the field are returned on a read from the terminal whether or not the application user modified the field.

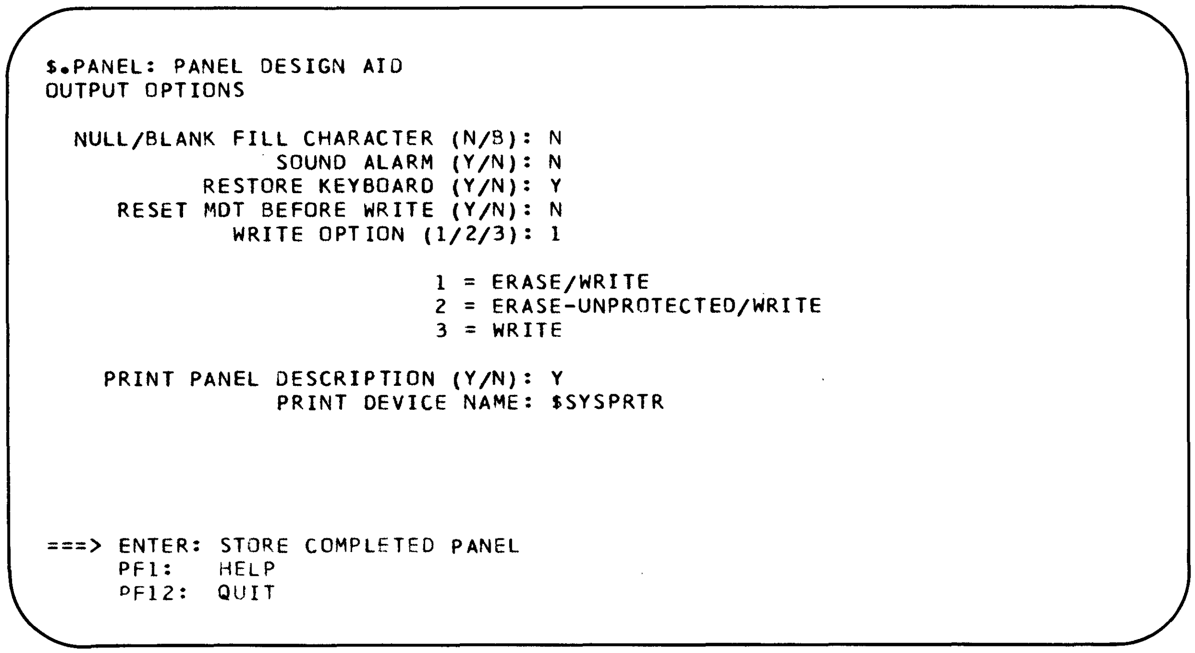
Selector-Pen Detectable:
If you specify "Y", the attribute for a detectable field is set. There are additional requirements on the field's content, length, and screen location, as described in the *3270 Description and Programmer's Guide.* Note that a nondisplay field cannot be detectable and that a bright field is always detectable (provided the other requirements are met).

Field Table Entry:
If you specify "Y", an entry for this field is placed in the field table. You should specify "Y" for fields that are modified or referenced by the application programs that use the panel.

Cursor Field:
If you specify "Y", the cursor is positioned at this field when the panel is used. If you specify more than one cursor field, the one you specify last takes effect. If you specify no cursor field, the cursor is positioned at the first unprotected field; if there are no unprotected fields, it is positioned at the first nonstandard protected field; if there are no nonstandard fields, it is positioned at row 1, column 1.

Output Options Phase

Output options is the last phase of panel design. A panel is not in the form that can be used by application programs until you complete this phase by pressing the ENTER key. The options used for the sample panel, as shown in Figure 13 , are the defaults.



```
$.PANEL: PANEL DESIGN AID
OUTPUT OPTIONS

  NULL/BLANK FILL CHARACTER (N/B): N
                SOUND ALARM (Y/N): N
           RESTORE KEYBOARD (Y/N): Y
     RESET MDT BEFORE WRITE (Y/N): N
             WRITE OPTION (1/2/3): 1

                           1 = ERASE/WRITE
                           2 = ERASE-UNPROTECTED/WRITE
                           3 = WRITE

    PRINT PANEL DESCRIPTION (Y/N): Y
                PRINT DEVICE NAME: $SYSPRTR



===> ENTER: STORE COMPLETED PANEL
     PF1:   HELP
     PF12:  QUIT
```

Figure 13. Screen Layout During Output Options Phase

The output options are:

Fill Character:
The specified character replaces trailing field fill indicators in the panel. Null allows the application user to use insert mode to enter characters in unused field positions; blank does not.

Sound Alarm:
If you specify "Y", the alarm is sounded when the panel is sent to the application user.

Restore Keyboard:
If you specify "Y", the keyboard is unlocked after the panel is sent to the user.

Reset MDT:
This option is relevant only for write options 2 and 3 (described next). If you specify "Y", modified data tags set as a result of the preceding read from the terminal are reset before the panel is sent to the user.

Write Options:
1 (erase/write) causes the panel to replace whatever was on the user's screen; 2 (erase-unprotected/write) and 3 (write) modify the contents of the screen. In addition, option 2 causes existing unprotected fields to be filled with nulls before the new fields are written.

Print Panel Description:
If you specify "Y", a description of the panel is printed on the device whose name you specify; $SYSPRTR is the default.

Figure 14 shows the printed description of the sample panel.

```
                                   DESCRIPTION OF PANEL CFMENU

                 PANEL NAME: CFMENU
                    DATASET: $.SYSPNL
                     VOLUME: EDX002

        FIRST RECORD NUMBER:   121
          NUMBER OF RECORDS:     9
           EFFECTIVE LENGTH:   600
    MINIMUM CF BUFFER SIZE:    768
                 PANEL SIZE:  1920 (24 X 80)

               WRITE OPTION: ERASE/WRITE
             FILL CHARACTER: NULL
                SOUND ALARM: NO
           RESTORE KEYBOARD: YES
    RESET MDT BEFORE WRITE: NO

    STANDARD FIELD INDICATOR: &
   PROTECTED FIELD INDICATOR: !
 UNPROTECTED FIELD INDICATOR: @
              FILL INDICATOR: /
```

Figure 14 (Part 1 of 3). Sample Print Panel Description

```
                           DESCRIPTION OF PANEL CFMENU

       5    10   15   20   25   30   35   40   45   50   55   60   65   70   75   80
   ----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
  | *******************************************************************************|
  |                                                                                |
  |             EVENT DRIVEN EXECUTIVE COMMUNICATIONS FACILITY                     |
  |                                                                                |
 5-                                SAMPLE PROGRAM                                  -
  |                                                                                |
  | *******************************************************************************|
  |                                                                                |
  |                                                                                |
10-                  CONNECT TO HOST: ////////   (ENTER HOST TERMINAL NAME)       -
  |                                                                                |
  |         CONNECT TO APPLICATION PGM: ////////   (ENTER PROGRAM NAME)            |
  |                                                                                |
  |             LOAD $FSEDIT UNDER EDX: ////////   (ENTER WORKFILE NAME)           |
15-                                                                                -
  |           DISCONNECT THIS TERMINAL: /          (ENTER ANY CHARACTER)           |
  |                                                                                |
  |                                                                                |
  |                                                                                |
20- ===> TAB TO FIELD AND ENTER DATA                                               -
  |                                                                                |
  |                                                                                |
  |                                                                                |
  |                                                                                |
   ----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
```

Figure 14 (Part 2 of 3). Sample Print Panel Description

```
                                  DESCRIPTION OF PANEL CFMENU

 FTAB    SCREEN LOCATION                ATTR BITS
 ENTRY  POSITION ROW COLM   LENGTH     2 3 4 5 7   CURSOR   CONTENT
 -----  -------- --- ----   ------     - - - - -   ------   -------
               1   1    2       79     1 0 0 0 0            ****************************************************************
             177   3   18       46     1 0 0 0 0            EVENT DRIVEN EXECUTIVE COMMUNICATIONS FACILITY
             353   5   34       14     1 0 0 0 0            SAMPLE PROGRAM
             481   7    2       79     1 0 0 0 0            ****************************************************************
             739  10   20       16     1 0 0 0 0            CONNECT TO HOST:
   1         756  10   37        8     0 0 0 0 0     **     ////////
             767  10   48       26     1 0 0 0 0            (ENTER HOST TERMINAL NAME)
             888  12    9       27     1 0 0 0 0            CONNECT TO APPLICATION PGM:
   2         916  12   37        8     0 0 0 0 0            ////////
             927  12   48       20     1 0 0 0 0            (ENTER PROGRAM NAME)
            1052  14   13       23     1 0 0 0 0            LOAD $FSEDIT UNDER EDX:
   3        1076  14   37        8     0 0 0 0 0            ////////
            1087  14   48       21     1 0 0 0 0            (ENTER WORKFILE NAME)
            1210  16   11       25     1 0 0 0 0            DISCONNECT THIS TERMINAL:
   4        1236  16   37        1     0 0 0 0 0            /
            1247  16   48       21     1 0 0 0 0            (ENTER ANY CHARACTER)
            1521  20    2       32     1 0 0 0 0            ===> TAB TO FIELD AND ENTER DATA
   5        1606  21    7        0     1 0 0 0 0




  FIELD ATTRIBUTES:  BIT   MEANING
                     ---   -------
                      2     0 = UNPROTECTED, 1 = PROTECTED
                      3     0 = ALPHAMERIC,  1 = NUMERIC (PROTECTED + NUMERIC = AUTOSKIP)
                     4&5   00 = NORMAL INTENSITY, NOT DETECTABLE
                           01 = NORMAL INTENSITY, DETECTABLE
                           10 = BRIGHT INTENSITY, DETECTABLE
                           11 = NONDISPLAY, NOT DETECTABLE
                      7    MODIFIED DATA TAG
```

Figure 14 (Part 3 of 3). Sample Print Panel Description

The first page gives general information about the panel. The first three items identify the panel. Then the location of the member, relative to the start of the data set, and size of the member appear. The effective length shows how many bytes of the member are actually used; this item and the next (buffer size) are discussed under "Using S$GETPNL to Fetch Panels" on page 61.

The second page shows the panel as the user sees it, except that field fill indicators are not replaced by blanks, and lowercase data prints as uppercase.

The rest of the report describes the fields that make up the panel. The first column shows the relative location of the field's entry in the field table, if any. The next four columns show the field's position and length. Next come the field's attributes, shown as they appear in a 3270 attribute character; an explanation of the bits is printed at the bottom of the last page of the report. Bits 0 and 1, which are set according to the value of the other bits, and bit 6, which is always 0, are not shown. The column headed CURSOR contains asterisks for the field at which the cursor is positioned when the panel is used.

Ending a Session with $.PANEL

When you press PF12 during any phase or ENTER during the output options phase, you get this message:

```
CONTINUE SESSION WITH $.PANEL? (Y/N):
```

If you respond "Y", you are returned to the first phase where you can specify the next panel that you want to work on. If you respond "N", your terminal is unlinked from $.PANEL, and you receive an END OF SESSION message. If no one else is using $.PANEL, the program is stopped.

$.PANEL User Messages

You may receive the following messages at your terminal while using $.PANEL. The messages are shown here in alphabetical order. Messages sent to the Communications Facility system log are shown in the *Operator's Guide.*

DATA SET DIRECTORY FULL
: There is no space in the partitioned data set directory for another member. If the data set contains deleted members, use the compress function of the EDX utility $DIUTIL to retrieve unused space. If not, allocate a new larger data set, copy the existing members to it, delete the old data set, and rename the new one. Do *not* do either of those steps while anyone else is using the data set.

DATA SET FULL
: There is no space in the partitioned data set itself for another member. If the data set contains deleted members, use the compress function of the EDX utility $DIUTIL to retrieve unused space. If not, allocate a new larger data set, copy the existing members to it, delete the old data set, and rename the new one. Do *not* do either of those steps while anyone else is using the data set.

DATA SET NOT FOUND
: The data set you specified does not exist. Enter the correct name, or allocate and initialize the data set that you want to use.

DISK I/O ERROR
An I/O error occurred while your panel was being read or written. The system log gives additional information.

DUPLICATE INDICATORS
The design indicator at which the cursor is positioned is not unique. Change it.

FIELD TOO LONG (>254)
The field at which the cursor is positioned is longer than the maximum allowed. You must correct it before you can obtain a panel display or complete the output options phase.

INDICATOR ALREADY USED AS PANEL DATA
The design indicator at which the cursor is positioned appears within the panel as field data. Specify a different indicator, or proceed to panel layout and change the panel data before again specifying this indicator.

INVALID OPTION
The option at which the cursor is positioned is invalid. Correct it.

MAXIMUM NUMBER OF USERS ACTIVE; PLEASE TRY AGAIN LATER
There is not enough dynamic storage for another user. If this problem occurs often, consider increasing the amount of storage as explained in the section "Using $.PANEL to Design Panels" on page 45.

MEMBER NOT BUILT BY $.PANEL
The member (panel) that you specified exists, but it does not appear to have been built by $.PANEL. This may be because of a disk I/O error, but it is more likely that you did not specify the name that you intended.

NO FIELDS DEFINED
The panel is completely empty. It must contain at least one field before you can obtain a panel display or complete the output options phase.

NO INITIAL FIELD START INDICATOR
The first nonblank character in the panel is not a field start indicator. You must correct this before you can obtain a panel display or complete the output options phase.

NO NON-STANDARD FIELDS DEFINED
This message indicates that the panel contains only standard fields, so there is nothing to do in the field attributes phase.

PANEL DESCRIPTION NOT PRINTED, NO SPACE TO LOAD PROGRAM
When you request a printed panel description, $.PANEL loads another program to do it, and there was not enough space in any partition. Repeat the request when there is less work in the system or use $.PNLUT1 to obtain a panel description.

PANEL DESCRIPTION NOT PRINTED, PRINT DEVICE NAME NOT DEFINED
The print device name that you specified is not known to EDX. Repeat the request with a correct name.

PANEL DESCRIPTION NOT PRINTED, PRINTER IS BUSY
The print device that you specified is busy. Repeat the request when the printer is available, specify a different printer, or use $.PNLUT1 to obtain a panel description.

PANEL DESCRIPTION NOT PRINTED, PROGRAM LOAD ERROR
When you request a printed panel description, $.PANEL loads another program, and the load failed. The system log gives additional information.

PANEL EXCEEDS MEMBER SIZE
The panel, in its final form, is too long to fit in the allocated member. This occurs only when a panel contains a great many, very short fields. Each field requires 5 bytes in the 3270 data stream, in addition to its contents. An unprotected field that is immediately followed by one or more undefined positions requires an additional 2 bytes. Modify the panel to reduce the size of the resulting data stream or to reduce the number of field table entries (each requires 4 bytes). If you cannot do this, delete the member, increase the member size that $.PANEL allocates (see "Changing Member Size" on page 46), and define the panel again.

PANEL IS IN USE
Another user of $.PANEL is working on the panel that you specified. Try again later.

PANEL *name* COMPLETED AND STORED
The panel is stored in the form that can be used by application programs.

PANEL *name* SAVED
The panel is saved in its design form. It can be used with $.PANEL but not by application programs.

REQUIRED PARAMETER NOT SPECIFIED
The parameter at which the cursor is positioned is blank. Enter a correct value.

TOO MANY NON-STANDARD FIELDS (>91)
You have specified more nonstandard fields than can be recorded in the allocated member. Reduce this number, perhaps by changing some protected fields to standard fields. If you cannot do this, delete the member, increase the member size that $.PANEL allocates (see "Changing Member Size" on page 46), and define the panel again.

VOLUME NOT FOUND
The volume you requested does not exist. Enter the correct name, or ready the volume that you want to use.

$.PANEL HAS BEEN STOPPED
The program has stopped because of an operator command or an error. The system log shows the reason.

Using S$GETPNL to Fetch Panels

S$GETPNL is a subroutine used in application programs to fetch panels defined by $.PANEL.

S$GETPNL Calling Sequence

The calling sequence for S$GETPNL is:

CALL S$GETPNL,*buffer,name,dscb,rcode*

where:

buffer
: is the address of a Communications Facility buffer to receive the panel.

name
: is the address of an 8-byte area that contains the panel name.

dscb
: is the address of the data set control block for the partitioned data set in which the panel is stored.

rcode
: is the label of a word to receive the return code.

S$GETPNL Output

S$GETPNL locates the panel in the data set and reads it into the buffer.

There are three sections in a panel:

- *Header information* that gives the length of the 3270 data stream and the relative location of its beginning and end.
- *Field table* (optional).
- *3270 data stream,* which produces the panel when it is sent to a Communications Facility terminal.

Header Information

The header information overlays words 2 to 5 of the buffer header, with the relative locations converted to actual buffer addresses.

On return from S$GETPNL, the buffer data area contains the field table and data stream, and the buffer header contains:

B$SIZE—Buffer size, not modified
B$COUNT—Length of 3270 data stream
B$ADDR—Address of start of 3270 data stream
B$DATA@—Address of end of 3270 data stream plus one
B$TXTCT—Equal to B$SIZE

The buffer header information is such that the buffer can be used as the subject of a SEND M instruction to send the panel to a user of the application program. It is also such that you can use the PUT instructions to modify the panel before sending it.

The buffer must be large enough to contain the field table and data stream, with additional space if the application program appends data to it. $.PANEL allocates nine records for a panel (unless you have changed the member size), because that is what is needed during the design of a panel. The panel in its final form usually requires far less. S$GETPNL reads only the records that contain the final output. The printed report that you can obtain from $.PANEL or $.PNLUT1 shows the length of the final output (the effective length) and the minimum buffer size (the effective length rounded to a multiple of 256). See Figure 14 on page 57.

If you append data to the buffer, you must be sure that the buffer is large enough to contain it. PUT instructions can overrun the buffer with no error indication, because B$COUNT does not include the length of the field table that precedes the 3270 data stream. The space required for each field that you append to the buffer is the data length, plus 3 bytes for an SBA (set buffer address) order, plus 2 bytes if you specify the field's attributes.

Field Table

The field table contains one 4-byte entry for each field specified by the panel's designer.

Byte	Contents
0-1	Field position (its screen location relative to 0)
2	Field length (0-254)
3	Field type (1 if the field is unprotected and numeric, otherwise 0)

You can use the field position in these ways:

- As the "position" operand of a GET F instruction used to retrieve a specific field from the user's input to the panel.
- To identify which field was obtained as the result of a sequential GET F instruction.
- As the "position" operand of a PUT F instruction used to modify the field's attributes and/or contents.

In the latter case, the PUT F instructions addressed to a buffer that contains a panel do not modify the existing 3270 data stream; they append fields to it. Each field in the data stream begins with an SBA (set buffer address) order that contains the screen location at which the following data is to appear. When a data stream is sent to a terminal, a later specification for a particular screen location overlays an earlier specification. It may also overlay following fields; be careful that your data is no longer than it should be.

An SBA order need not be followed by field data. You can use a PUT F instruction with a null text operand and an ATTR= operand to modify the attributes of a field without changing its contents. You can use a PUT F instruction with the text operand containing an insert cursor order (X'13') to reposition the cursor.

The field table makes it possible to modify panels with little or no modification of the application programs that use them. If the screen location of a field is changed, but its relative location in the field table is not, no program change is required. If you expect a panel design to be volatile, programs should use symbolic names for field table entries. That makes it simple to modify programs so that, for example, the name QUANTITY now refers to the fifth entry instead of the third entry.

You can use the third byte of a field table entry (field length) in application programs that are not dependent on predefined field lengths. You can use the fourth byte (field type) to decide whether a GET F instruction is to include the TYPE=NUMERIC operand.

$.CFMENU, the sample program, illustrates the use of a field table. The program uses the screen locations from the field table to determine which option the user selected and to send error messages to the user.

Return Codes

S$GETPNL returns one of the following values in the *rcode* operand:

-1	Successful.
1-11	A disk read error, as described for the READ instruction in the *Event Driven Executive Messages and Codes*, occurred.
20	Member (panel) not found.
21	Member is in its design format, not its final format.
22	Member not built by $.PANEL (member code is not 12, or content is not that produced by $.PANEL).
23	Buffer too small.

You can use the LA command of the EDX $DIUTIL utility to determine whether a member was built by $.PANEL and whether it is in its design format or final format. For members built by $.PANEL, the member code is 12. For design format, the user code is 0. For final format, the user code is the effective length rounded to a multiple of 256 (the value reported by $.PANEL as the minimum buffer size).

Sample Application Program

Figure 15 on page 65 shows the skeleton of an application program that uses S$GETPNL. It illustrates the concepts discussed in the previous sections of this chapter and some additional requirements.

The data set referenced in the call to S$GETPNL must be open. The DS= operand of the PROGRAM statement accomplishes this.

The program must contain the Communications Facility equates (S$CFEQU), the EDX data set control block equates (DSCBEQU), and the panel fetch subroutine (S$GETPNL).

You can use the same buffer for terminal input and output, if you wish. After the panel is fetched, the buffer header field B$ADDR contains the address of the 3270 data stream. If the RECEIVE instruction uses the same buffer, the panel data stream is overlaid, but the field table at the beginning of the buffer is left intact.

If you use the same buffer for input and output, you must be sure that the terminal input will be no longer than the buffer size minus the length of the field table. The RECEIVE instruction can overrun the buffer with no error indication, because B$ADDR does not address the start of the buffer data area.

The MOVEA instruction resets buffer header field B$ADDR to address the start of the buffer data area, as it normally does. You may need to do this if you use the buffer for purposes other than processing panels.

Refer to Appendix A, "$.CFMENU Sample Program" on page 197 for another example.

Using Panels in Other Ways

You may write applications that need to access the output of $.PANEL other than through S$GETPNL. For such applications, S$PNEQU2 is provided. This component of the Communications Facility is a set of equates that defines the format of the partitioned data set member built by $.PANEL.

```
TASKNAME  PROGRAM   START,DS=name
          COPY      S$CFEQU                       CF EQUATES
          COPY      DSCBEQU                       EDX DSCB EQUATES
          COPY      S$GETPNL                      PANEL FETCH SUBROUTINE
BUFF      DEFINE    BUFFER,SIZE=nnn               I/O BUFFER
FLD1      EQU       BUFF                          FIRST FIELD TABLE ENTRY
FLD2      EQU       FLD1+4                        SECOND FIELD TABLE ENTRY
          •
          •
          •
PNLNAME   DATA      8C''                          PANEL NAME
RCODE     DATA      F'0'                          RETURN CODE
ORIGIN    TEXT      LENGTH=10                     USER'S STATION NAME AND TYPE
INPUT     TEXT      LENGTH=nn                     INPUT TEXT AREA
          •
          •
          •
START     EQU       *
          •
          •
          •
          MOVE      PNLNAME,name,(8,BYTE)         SET PANEL NAME
          CALL      S$GETPNL,(BUFF),(PNLNAME),(DS1),RCODE   FETCH PANEL
          IF        (RCODE,NE,-1),GOTO,error1     CHECK RETURN CODE
          PUT       FIELD,BUFF,text,FLD1*         MODIFY FIRST FIELD
          SEND      MESSAGE,ORIGIN,BUFF           SEND PANEL TO USER
          •
          •
          •
          RECEIVE   MESSAGE,BUFF,WAIT=YES,ORIGIN=ORIGIN    GET USER'S INPUT
          GET       FIELD,INPUT,BUFF,FLD1*        FIRST FIELD MODIFIED?
          IF        (INPUT-1,NE,0,BYTE)           YES (TEXT COUNT > 0)
          •
          •
          •
          MOVEA     BUFF+B$ADDR,BUFF              RESET BUFFER DATA ADDRESS
          •
          •
          •
```

Figure 15. Sample S$GETPNL Application

Using $.PNLUT1 to Print Panel Descriptions

$.PNLUT1 is a utility program that prints descriptions of panels created by $.PANEL. The panel descriptions are the same as those that you can obtain when you design panels. You can request a description of one panel, of a group of panels whose names begin with the same 1 to 7 characters, or of all panels in a data set.

The Communications Facility does not have to be active when you use $.PNLUT1. Use the EDX $L command to load the program into a mapped partition, specifying the name and volume of the partitioned data set that contains the panels. For example, you can enter:

```
$L $.PNLUT1 USERPNLS,EDX003
```

Then enter one of the following commands:

PM (Print 1 Member)
This command prints a description of the member whose name you specify.

PG (Print Generic Group of Members)
This command prints a description of each member whose name begins with the string of 1 to 7 characters you specify.

PALL (Print All Members)
This command prints a description of all members in the data set.

EN (End Program)
This command ends the program.

You will be prompted for a print device name after you enter the first print command.

Creating a Transaction

You can create a new transaction and specify what the program dispatcher is to do when a user enters that transaction from a terminal or a program. To create a transaction and integrate it into the Communications Facility system, you have to:

- Name the transaction (what the operator or programmer will enter to identify the transaction).

- Determine what will happen when an operator or programmer issues the transaction. Possible actions taken when a transaction is issued are:

  - Load a program somewhere in the network.
  - Create a station (and its associated message queue).
  - Send a message (the transaction the user entered) to the station so created.

  Assuming a program is loaded in response to the transaction, you have to decide whether that program will process one transaction and then terminate; process transactions until no more are available and then terminate; or remain in storage, waiting for transactions, until its station is explicitly stopped.

- Determine the transaction's *type*—one of 16 predefined categories. The transaction's type defines what is to happen when the transaction is entered.

- Code the transaction-processing program.

- Make an entry for the transaction in the program dispatcher's data set, $.SYSPD. The entry indicates the transaction's name and type, the name of the associated transaction-processing program, and other attributes of the transaction.

- Optionally, define the station created for your program in $.SYSNET.

"Creating a Program to Communicate with EDX Terminals" on page 85 includes a sample transaction-processing application. It illustrates many of the concepts presented in this chapter.

Naming the Transaction

The transaction name, or identifier, can be from 1 to 4 characters. It must be unique within the cell where the transaction is to be processed; it must not duplicate the names of IBM-supplied transactions, which are listed in the *Operator's Guide.*

Determining the Transaction's Type

The transaction type, which you specify when you define the transaction in $.SYSPD, tells the program dispatcher what to do when it receives the transaction. The transaction type consists of 2 digits. The first digit indicates the type of program that processes the transaction; the second indicates the various actions the program dispatcher may take when it receives the transaction. It's your responsibility to be sure your program is actually of the type you specify in the first digit.

The first digit can have these values:

1 A single-transaction program. This type of program processes a single transaction and terminates.

2 A multi-transaction program. This type of program processes transactions until there are no more and then terminates. The indicator that there are no more transactions depends on how you design your program. It may, for example, be that there are no more messages on the queue; a special transaction that contains an "end-of-data" indicator; or a request to stop.

3 A continuous program. This type of program remains in storage, waiting for transactions, until it is requested to stop.

4 A continuous, reentrant program. This type of program also remains in storage, waiting for transactions, until it is requested to stop.

The program dispatcher loads a type 1 program (if the second digit of the type code specifies so) each time the transaction is issued; it loads the other types only if the program is not already in storage. The program dispatcher may tell a type 3 or 4 program to stop when storage is needed for another program; it does so only if you specified that the program can be purged when you defined the transaction in $.SYSPD. To the program dispatcher, there's no difference between types 3 and 4. The two types are provided for your own documentation purposes.

The second digit can have these values:

0 Load a program; do not create a station; do not put the transaction on any queue. This type of transaction simply starts a program somewhere in the network.

1 Load a program; create a station and its associated message queue; do not put the transaction message on the queue.

2 Load a program; create a station and its associated message queue; put the transaction message on the queue.

3 Create a station and its associated message queue; put the transaction message on the queue; do not load a program. In this case, you must be planning to manage the message queue by some means other than having a program identified with the station.

Figure 16 on page 69 summarizes the transaction types.

Transaction Type	Load Single Transaction Program	Load Multi Transaction Program	Load Continuous Program	Load Reentrant Continuous program	Create Station and Queue	Put Transaction On Queue
10	Y	N	N	N	N	N
11	Y	N	N	N	Y	N
12	Y	N	N	N	Y	Y
13	N	N	N	N	Y	Y
20	N	Y	N	N	N	N
21	N	Y	N	N	Y	N
22	N	Y	N	N	Y	Y
23	N	N	N	N	Y	Y
30	N	N	Y	N	N	N
31	N	N	Y	N	Y	N
32	N	N	Y	N	Y	Y
33	N	N	N	N	Y	Y
40	N	N	N	Y	N	N
41	N	N	N	Y	Y	N
42	N	N	N	Y	Y	Y
43	N	N	N	N	Y	Y

Figure 16. Transaction Types

Coding the Transaction-Processing Program

When you're coding the transaction-processing program, you need to take into account various considerations that depend on the transaction type. This section explains those considerations.

In the examples that follow, the instruction:

```
IF (END,EQ,YES)
```

represents a test for whatever termination condition your program is designed to recognize.

Type 10 Transaction

With transaction type 10, there is no station to manage and no transaction to receive. A new copy of the program is loaded each time the transaction is issued. This type is appropriate for a batch program.

Type 11 Transaction

A new copy of the program is loaded each time the transaction is issued. A message type station with the same name as the program is created if it doesn't already exist, but the transaction message isn't put on the queue. You're responsible for deleting the station.

This is an example of a type 11 transaction-processing program:

```
PROG11      PROGRAM     START
START       •
            •
            Program functions
            •
            •
            Delete station and terminate
```

Type 12 Transaction

A new copy of the program is loaded each time the transaction is issued. A message type station with the same name as the program is created if it doesn't already exist, and the transaction message is put on the queue. You're responsible for handling the message and deleting the station. Transaction type 12 would be appropriate, for example, for a program that simply displays a menu at the terminal.

This is an example of a type 12 transaction-processing program:

```
PROG12      PROGRAM     START
START       •
            •
            RECEIVE     M,BUFF
            •
            •
            Process the transaction
            •
            •
            Delete station and terminate
```

Type 13, 23, 33, or 43 Transaction

A message type station with the same name as the program is created if it doesn't already exist, and the transaction message is sent to that station, but no program is loaded. You're responsible for handling the messages and deleting the station. These types may be used, for example, to accumulate a queue of transaction messages for later processing. When it's time to process the transactions, use the EDX $L command to load the program.

Type 20, 30, or 40 Transaction

The program is loaded if it isn't already in storage. There is no station to manage and no transaction to receive. These transaction types are appropriate for batch programs.

Type 21 Transaction

The program is loaded if it isn't already in storage. A message type station with the same name as the program is created if it doesn't already exist, but the transaction message isn't put on the queue. You're responsible for deleting the station.

This is an example of a type 21 transaction-processing program:

```
PROG21     PROGRAM     START
START      •
           •
           Program functions
           •
           •
           IF          (END,EQ,YES)
             Delete station and terminate
           ENDIF
           •
           •
           GOTO        START
```

Type 22 Transaction

The program is loaded if it isn't already in storage. A message type station with the same name as the program is created if it doesn't already exist, and the transaction message is put on the queue. You're responsible for handling the message and deleting the station.

This is an example of a type 22 transaction-processing program:

```
PROG22     PROGRAM     START
START      •
           •
           RECEIVE     M,BUFF
           •
           •
           Process the transaction
           •
           •
           IF          (END,EQ,YES)
             Delete station and terminate
           ENDIF
           •
           •
           GOTO        START
```

Type 31 or 41 Transaction

The program is loaded if it isn't already in storage. A message type station with the same name as the program is created if it doesn't already exist, but the transaction message isn't put on the queue. You're responsible for managing and deleting the station. Your program must check for a Communications Facility status message and stop when it receives one.

This is an example of a type 31 or 41 transaction-processing program:

```
PROG3141     PROGRAM     START
START        •
             •
             Program functions
             •
             •
             RECEIVE     M,BUFF,WAIT=NO
             IF          (PROG3141,EQ,6)
               Delete station and terminate
             ENDIF
             •
             •
             GOTO        START
```

Type 32 or 42 Transaction

The program is loaded if it isn't already in storage. A message-type station with the same name as the program is created if it doesn't already exist, and the transaction message is put on the queue. You're responsible for handling the message and deleting the station. Your program must check for a Communications Facility status message and stop when it receives one.

This is an example of a type 32 or 42 transaction-processing program:

```
PROG3242    PROGRAM    START
START       •
            •
            RECEIVE    M,BUFF
            IF         (PROG3242,EQ,6)
              Delete station and terminate
            ENDIF
            •
            •
            Process the transaction
            •
            •
            GOTO       START
```

Terminating Your Program

A single-transaction program (type 1*n*) terminates when it has finished its work. A multi-transaction program (type 2*n*) terminates when it has finished its work or when it has been requested to stop. A continuous program (type 3*n* or 4*n*) terminates when it has been requested to stop.

Your program receives the request to stop in the form of a status message, which is sent as a result of a halt command for the program dispatcher or a stop or halt command for your program. If you defined your program as purgable, it is also requested to stop when its storage is needed for another transaction-processing program.

If you want, you can determine why your program was told to stop by checking whether its station block is active (bit 0 of Q$STAT, 1=active, 0=inactive) and examining the first character of the status message:

- Your program was stopped—station block is inactive, status message starts with P.
- Your program was halted—station block is inactive, status message starts with H.
- Storage is needed for another program—station block is active, status message starts with P.
- Program dispatcher was halted—station block is active, status message starts with H.

How you terminate your program depends partly on the transaction type and partly on the characteristics of your transaction-processing system. You need to consider what you know about your system as you apply the suggestions in this section.

Also consider that the program dispatcher and your program are operating asynchronously. Their functions can conflict if you code your program incorrectly. The two things to keep in mind are:

- $.PD may load your program if it isn't already in storage; you terminate the program.
- $.PD may create a station if it doesn't already exist; you delete the station.

When there is no station to manage (type 10, 20, 30, or 40), just terminate the program when it finishes its work:

```
PROGSTOP LOGMSG=NO
```

When there is a station, but transactions aren't sent to it (type 11, 21, 31, or 41), you can delete the station as you terminate the program:

```
LOCATE ST,#1,OPTION=PROGSTOP
```

When your program processes a single transaction (type 12 or 13), you are responsible for deleting the station, but you can't do so unconditionally. By the time you've finished processing the transaction, $.PD may have received and sent another one and loaded another copy of the program. If the station has no disk queue, use the DELETE option of the LOCATE instruction, which will delete the station only if it has no pending storage-queued messages:

```
LOCATE   ST,#1,OPTION=DELETE
PROGSTOP LOGMSG=NO
```

If the station has a disk queue, use the RECEIVE N instruction to check for messages before deleting the station:

```
RECEIVE  N,WAIT=NO
IF       (taskname,EQ,1)
  LOCATE ST,#1,OPTION=DELETE
ENDIF
PROGSTOP LOGMSG=NO
```

When your program processes multiple transactions until there are no more (type 22 or 23), you need to consider what the "no more" indicator is. If it's an empty queue and the station has no disk queue, you can delete the station conditionally and proceed according to the return code:

```
LOCATE   ST,#1,OPTION=DELETE
IF       (taskname,EQ,5),GOTO,receive-transaction
PROGSTOP LOGMSG=NO
```

As an alternative, or if the station has disk queue, use the RECEIVE N instruction instead:

```
RECEIVE  N,WAIT=NO
IF       (taskname,LT,0),GOTO,receive-transaction
LOCATE   ST,#1,OPTION=PROGSTOP
```

When your program processes transactions until it's requested to stop (type 32, 33, 42, or 43) and you didn't define it as purgable, a stop request means that it's time to shut down. You can, if you wish, receive and process any pending transactions. Then delete the station and terminate.

If you did define your program as purgable, you may want to handle a purge request differently from other stop requests. When you receive a purge request, you could ignore pending transactions and terminate without deleting the station. You should do this only if you know that more of your program's transactions will be issued. If this isn't the case, and there are pending transactions when you receive the purge request, your program may never be reloaded to process them.

In any case, be sure that you do nothing between detecting an empty queue and terminating your program. For example, if you issued a log message before terminating, the following sequence of events could occur:

1. Your program detects that its queue is empty and deletes its station.

2. Your program issues a log message.

3. While your program is waiting for the SEND LOG instruction to complete, $.PD receives another transaction for your program. It creates the station and sends the transaction. It doesn't load the program, because it's already in storage.

4. The SEND LOG instruction completes, and your program terminates.

As a result, the pending transaction won't be processed unless yet another transaction for your program is issued.

Finally, you need to decide whether or not to close your station's disk queue, if it has one, before terminating your program. A station's disk queue is closed when the station is stopped or halted; deleting a station block doesn't affect the disk queue. You don't have to close the disk queue before terminating. If you leave it open, it will be available when the station is started again. If you prefer to close the disk queue, issue a SEND CP instruction, specifying a stop or halt command for your station and ACK=YES. You can delete the station block and terminate when the instruction completes, or you can wait until you receive the status message that results from the stop or halt command.

Using Transactions to Segment Your Program

By dividing your application into several programs that respond to different transactions, you can effect substantial storage savings by having only one transaction-processing program in storage at any time.

This section and the next discuss how to do this, using a work session controller application as an example. You can apply the concepts to any transaction-processing application. The work session controller transactions are described in "Creating a Program to Communicate with EDX Terminals" on page 85.

For example, say your application displays a menu on the terminal and waits for the operator to select an option. The program might issue these work session controller transaction commands:

SD—Start device
SS—Start session
BI—Send image
WK—Wait for key
RD—Read data
LI—Link to program

You might split your application into three small programs:

PROG1	Issue SD and SS SD *Start device* SS *Start session with secondary transaction ID PROG2*
PROG2	Issue BI and WK BI *Send image* WK *Wait for key with secondary transaction ID PROG3*
PROG3	Issue RD and LI RD *Read data from the screen* LI *Link to PROG2 if input is invalid; else link to requested program*

None of these programs have to be in storage while the operator is deciding which option to select. You could code these programs as type 22 programs, terminating each after it issues its second transaction. When the operator makes a selection, PROG3 will be loaded. If the operator's input is invalid, PROG2 will be loaded.

Saving Time Instead of Storage

Although the program dispatcher uses a high-speed loader, the time required to load type 22 programs repeatedly may have an unacceptable effect on performance. If so, you can code your interactive programs as type 32 programs, which remain in storage until they are requested to stop. Using the example from the preceding section ("Using Transactions to Segment Your Program" on page 74), PROG2 and PROG3 could each be coded as a type 32 program, or their functions could be combined into a single type 32 program.

If you require optimum performance, but storage is sometimes at a premium, you should define type 32 programs as purgable. If you do so, the program dispatcher may request the program to stop when it needs storage for some other transaction-processing program. When the program is purged, you must ensure that it will be reloaded to continue its work. You can do this by specifying your program's transaction identifier as the secondary transaction identifier in each WSC transaction that it sends. After the program is purged, an acknowledgment transaction sent by the work session controller will cause it to be reloaded. In contrast, a type 22 program (or any type not defined as purgable) need request an acknowledgment only when the acknowledgment contains information the program requires.

Exchanging Transactions with Communications Facility Terminals

You may want to write an application program that processes transactions received from a Communications Facility terminal. This section explains how your program receives transactions from a Communications Facility terminal and how it can send a response to the terminal.

Receiving Transactions from the Terminal

When you define the transaction, you specify, in its transaction identifier statement, that data received as a result of this transaction will be in the form of a 3270 data stream. You must link the terminal to the program dispatcher, $.PD. For example, say that you have a program named PROG that is designed to process transaction TRID from a 3270-type terminal, as shown in Figure 17 on page 77.

The terminal's station is linked to the program dispatcher through this command:

```
CP LINK 3270ST $.PD
```

Transaction TRID must be defined to the program dispatcher by a TID statement in data set $.SYSPD. For example:

```
TID TRID PROG 32,,S
```

The last operand (S) specifies that the transaction is a 3270 data stream that the program dispatcher is to route without modification.

The terminal operator must enter the transaction identifier (in this case, TRID) as the first field on the screen. When the terminal operator enters a transaction, the data stream is sent to the program dispatcher.

If the first field of the data stream is TRID, the program dispatcher sends the entire data stream (including the 3270 header at the beginning and the ETX at the end) to your program, PROG.

If the first field of the data stream isn't TRID (or some other transaction identifier defined to the program dispatcher), the program dispatcher sends the message to the sender's alternate link, if one is defined and active; otherwise, it sends the message to $.WASTE as undeliverable. Note that the data stream created when the operator presses CLEAR or a PA key (a short read sequence) will not begin with a transaction identifier; those messages will go to the alternate link or to $.WASTE, not to your program.

In example 1, station 3270ST has no alternate link, so undefined transactions and short read data streams are sent to $.WASTE.

A second example is shown in Figure 18 on page 78.

The CP LINK command to link the terminal to the program dispatcher and the TID statement are the same as in example 1. In this example, an alternate link (ALTST) is defined for station 3270ST:

```
CP LINK 3270ST ALTST ALT
```

In example 2, undefined transactions and short read data streams are routed to ALTST. Figure 18 on page 78 shows ALTST as a station associated with a BSC

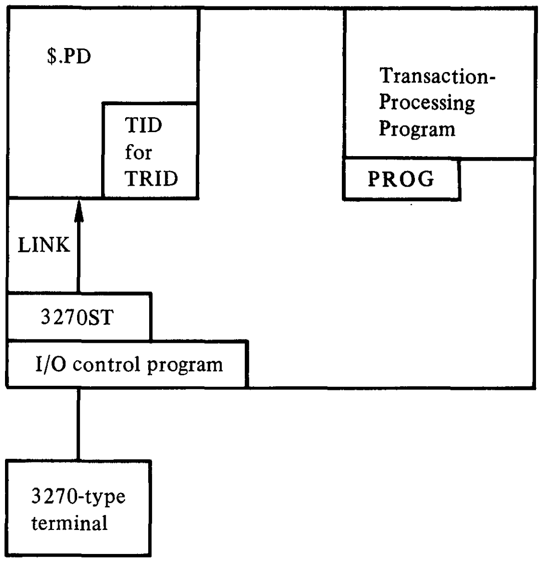


Figure 17. 3270 Transaction Processing—Example 1

line and illustrates a way of routing transactions to a host transaction-processing program. With this setup, transaction TRID and any other transactions defined to the program dispatcher are routed to local programs. All undefined transactions are routed to the host.

A third example is shown in Figure 19 on page 79.

In this example, the transaction-processing program is itself the alternate link for the terminal:

```
CP LINK 3270ST PROG ALT
```

With this setup, all input from the terminal, including undefined transactions and short read data streams, goes to program PROG.

Sending a Response to the Terminal

You may want to write a program that sends a response to a 3270-type terminal after receiving a transaction from it. You can send the response either as an ordinary data message or as a transaction.

To send the response as a data message, use the ORIGIN operand on the instruction that receives transactions, and send the response to the origin station. Format the response as a 3270 data stream. If your application program decides which transaction the operator will issue next, include the transaction identifier in the data stream as the first screen field. The attributes of the field should be protected (so that the operator can't modify it), nondisplay (because the operator doesn't need to see it), and MDT. The MDT (modified data tag) attribute will cause the transaction identifier to be included as the first field of the next input data stream.

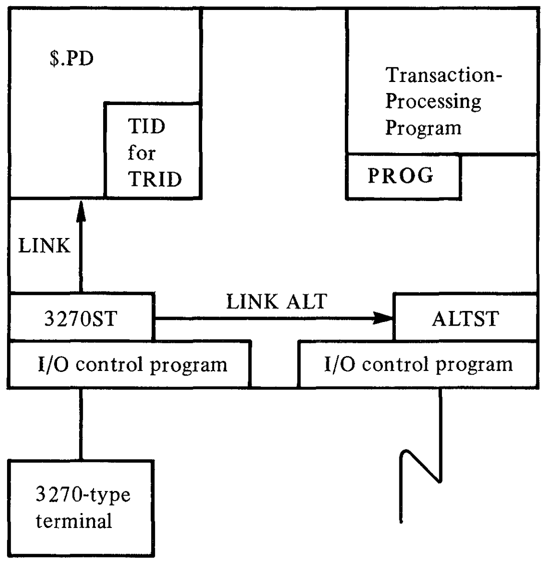


Figure 18. 3270 Transaction Processing—Example 2

To send the response as a transaction, you must define the terminal station to the program dispatcher as if it were a transaction-processing program. For example, to define transaction identifier T37A for station 3270ST:

```
TID T37A 3270ST,,,N 13,,S
```

Notice the type code, 13. This type code does *not* cause the program dispatcher to try to load a program named 3270ST.

Your program must have a way of associating a transaction identifier with a terminal. You could have a table that equates transaction identifiers to station names, or you could have the operator specify the transaction identifier as part of the initial transaction. Whichever technique you use, you send a response to station 3270ST by sending a T37A transaction to the program dispatcher. Make transaction identifier T37A the first field in the data stream; it can be at any screen location. The attributes of the field should be protected and nondisplay. Set the MDT bit if you want the field to be included in the next input data stream. You can include the identifier of the transaction the operator is to issue next as the first screen field, as already discussed.

The following instructions build a data stream that contains an erase/write command, T37A as the first field, and TRID as the second field:

```
PUT  CO,BUFF,OPTION=ERASE
PUT  F,BUFF,'T37A',ROW=1,COLM=2
PUT  F,BUFF,'TRID',ROW=1,COLM=2,ATTR=X'6D'
```

T37A is in the data stream to cause the program dispatcher to route the transaction to station 3270ST. It will not be written to the terminal, because it will be overlaid by the second field which has the same screen location.

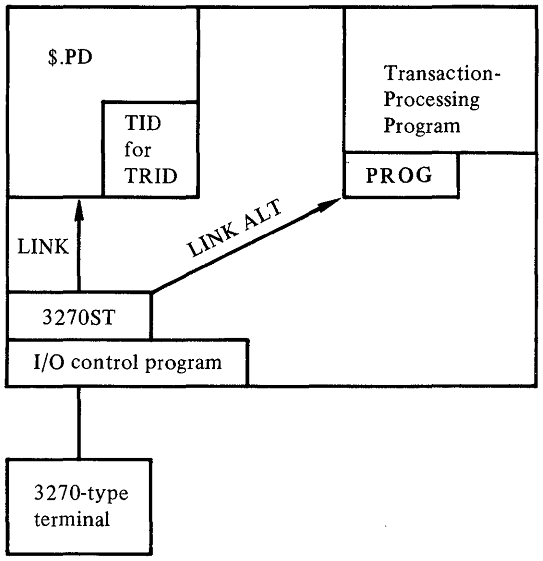


Figure 19. 3270 Transaction Processing—Example 3

Communicating with Remote Terminals

All the examples in this section have the terminal and the transaction-processing program in the same cell, so no cell identifier is required. When the first field of the data stream is 4 bytes, the program dispatcher assumes that the transaction is to be processed in the local cell. If you need to route data stream transactions to remote cells, include the cell identifier as the fifth and sixth bytes of the first field.

Entering the Transaction into $.SYSPD

The final step in creating a transaction is to make an entry for it in the $.SYSPD data set.

Use an EDX editor to add the following TID (transaction identifier) statement to the data set:

TID *tranid pgm*[,*vol,part,prefind*][*type*,P,S,R][H]

tranid
is the 1- to 4-character transaction identifier.

pgm
is the 1- to 8-character name of the program that is to process the transaction.

vol
is the 1- to 6-character volume name of the program. The default is the IPL volume.

part
is the number of the partition where the program is to be loaded. You can specify any of the following:

1 to 8
means that the specified mapped partition is used. If the partition does not exist or is not mapped, the partition specification in the transaction table is set to 0.

0
means that any available mapped partition is used.

-1 to -8
means that any available mapped partition is used except the one specified.

CF
means that the $.CF partition is used.

NCF
means that any mapped partition is used except the $.CF partition.

prefind
indicates whether or not a prefind of the program's data sets and overlays is to be performed when the program dispatcher is started. Omit the parameter to enable prefind; specify N to disable prefind.

type
is the transaction type, discussed under "Determining the Transaction's Type" on page 67.

P
indicates that the program may be purged to make its storage available to other transaction-processing programs. This parameter is valid only for transaction types 30 to 43.

S
indicates that this transaction is a 3270 data stream that the program dispatcher is not to modify.

R
indicates that the program load retry counts set by the PD RC command are to be ignored for this transaction. When there is no storage available for the program, the load is to be retried until it succeeds or until the program dispatcher is shut down.

H
puts the transaction on a hold queue. You must use a PD modify command or transaction to release it.

Defining the Station in $.SYSNET

If you want the station that the program dispatcher creates for your program to have a disk queue, or if you want its type to be other than message, you must define that station in $.SYSNET. If you want your program's transaction messages to be queued in storage to a message-type station, there is no need to define the station.

When the station is defined in $.SYSNET, the program dispatcher issues a CP S (start) command for the station instead of issuing a LOCATE instruction to create the station block. If the station type is USER, the program is loaded by the start processor, and it may not be loaded into the partition you specified in the TID statement. If you specified the partition as 1 to 8, an attempt is made to load the program into that partition. If there isn't enough space, or if you specified some other value for *partition*, the program is loaded into any available mapped partition. To ensure that the program is loaded into the partition you specify, define the station type as MSG so that the program will be loaded by the program dispatcher.

To define a message station with a disk queue, either use the $.CONFIG utility program or issue the following CP command in the cell where your program runs:

>CP DEF *name na* MSG *dsname*[,*volume*] {Y | N}

name
: is the 1- to 8-character name of the transaction-processing program.

na
: is the 4-digit hexadecimal network address, as appropriate to your network configuration.

dsname
: is the 1- to 8-character name of the disk-queue data set.

volume
: is the 1- to 6-character name of the volume where the disk-queue data set resides. The default is the volume where the message dispatcher resides.

Y | N
: indicates whether or not disk queuing is to be activated when the station is started.

Tracing Transactions

While you're developing a program or doing debugging, you may want to have a record of all transactions processed by the program dispatcher. To get such a trace, use the PD TRAC command. When you enter the command, you can direct the trace output to any EDX terminal or to any station in a Series/1 where the program dispatcher resides. You might choose a station whose messages are disk-queued, thereby putting the trace output on disk for later display.

One line is displayed for each transaction received or sent by the program dispatcher. These are some sample lines from the trace output:

```
2272 IN $.HMU      0016  SYSTSJHMU SJLU$.RMU
 205 IN $.PD       0017  HMU SJSYST??LU$.RMU   N
  18 OT $.HMU      0017  HMU SJSYSTSJLU$.RMU   N
```

The first item in each line is the time that has elapsed, in milliseconds, since the preceding trace output. The second item indicates whether the program dispatcher sent (OT) or received (IN) the transaction. The third item is the name of the station from which the transaction was received or to which it was sent. The fourth item is the length of the transaction, in hexadecimal. The remaining data is the transaction itself—the first 50 bytes for a character display, and the first 40 bytes for a hexadecimal display.

The format of the PD TRAC command is:

PD TRAC	**{ON | OFF}** [{*destination* | *}] **[X]**

ON | OFF
turns the trace on or off.

destination
is the name of the EDX device or the station to receive the trace output.

*
displays the trace on the terminal where the PD TRAC command was entered.

X
indicates that the trace output is to be hexadecimal; if X is omitted, the trace output is character.

Testing Transactions

While you're testing a transaction-processing program, you may want to execute it under control of the EDX $DEBUG program. To do so, you run the program in test mode. You can run a program in test mode only if it is loaded by the program dispatcher; do not define a user station for the program.

For example, say you're debugging program XYZPROG, which processes transaction XYZ. Transaction XYZ has this transaction identifier:

```
TID XYZ XYZPROG,,3 22
```

First you use the PD F command to place the transaction in test mode:

```
> PD F TID XYZ TE ON
```

Putting the transaction in test mode means that when the transaction is entered, the program will be loaded but not executed.

Now enter an XYZ transaction, thereby loading the program:

```
> TRAN XYZb00bbbbbboptional data
```

Load $DEBUG and set whatever breakpoints you want. Then issue this command:

```
> GOTEST
```

You'll get the message:

```
START XYZPROG (Y/N)?
```

Enter Y to start execution of XYZPROG, and debug the program.

You can run several programs under control of $DEBUG at the same time, but you must load and activate the programs one at a time. For example, you are debugging XYZPROG and ABCPROG, and you have entered a transaction to load XYZPROG. If you then enter a transaction to load ABCPROG, you'll get these log messages:

```
PD41 I 0000 $.PD XYZPROG WAITING, MUST BE STARTED FIRST
PD22 E 0000 $.PD ABC DISCARDED
```

Issue a GOTEST command to start execution of XYZPROG. Then re-enter the transaction to load ABCPROG, and issue a GOTEST command for it.

To remove the test mode option, enter:

```
> PD F TID XYZ TE OFF
```

For the complete syntax of PD F and TRAN, see the *Operator's Guide*.

Creating a Program to Communicate with EDX Terminals

The Communications Facility allows you to write interactive application programs that communicate with EDX terminals anywhere in the network. Such programs can communicate with:

- 3101, 4978, 4979, and 4980 terminals, and
- 4973, 4974, 4975, 5219, 5224, and 5225 printers.

The program can communicate with multiple terminals, which can be attached to any Series/1 in the network. The terminals need not be defined to the Communications Facility, but they must be defined to EDX.

The Communications Facility program that allows you to manage such terminals is called the *work session controller.* You communicate with it by means of transactions. You send the work session controller a transaction specifying what you want done at the terminal—for example, reading data, writing data, or sounding a tone. The work session controller, running in the Series/1 to which the terminal is physically attached, uses EDX I/O instructions to perform the function you requested. It sends another transaction as an acknowledgment.

You can also communicate with the work session controller through a set of subroutines that can be called from COBOL or EDL programs. The subroutines provide an interface similar to that of the EDX Multiple Terminal Manager (Program Number 5719-MS2). Most Multiple Terminal Manager applications can be converted to run under the Communications Facility. The subroutines are discussed in the *Communications Facility Work Session Controller High-Level Language Subroutines Programmer's Guide.*

Images displayed through the work session controller are built through $IMAGE and stored in the partitioned data set $.WSCIMG. "Storing Images in the Image Library" on page 89 explains how to convert $IMAGE images to work session controller images.

The Communications Facility also includes a program, $.WSMENU, that you can use to start communication between your application and its users. "Using $.WSMENU" on page 90 describes $.WSMENU.

The sample application in this chapter illustrates the use of some work session controller transactions.

Sending WSC Transactions

To make an I/O request, you send a transaction whose transaction ID is WSC. The WSC transaction, like any other transaction, is a fixed-format message. Part of the fixed format is a 2-character command in bytes 13-14. You use the commands to specify what you want done at the terminal. The commands are described under "Work Session Controller Transaction Commands" on page 87.

In your program, you use the SEND MT or SEND TT instruction to send a WSC transaction, just as you send any other transaction. The transactions are of various lengths; be sure you set the correct count in the text header or the buffer header.

WSC Acknowledgment Transactions

When you send a WSC transaction, the work session controller does what you requested; it may then send an acknowledgment transaction. For some WSC transaction commands, an acknowledgment transaction is required and you must specify where it is to be sent; in those cases, the acknowledgment transaction contains data or a return code. For other commands, the acknowledgment transaction is sent only if you request it; in those cases, it simply indicates that the function you requested was performed.

To specify where the acknowledgment transaction is to be sent, you use the secondary transaction and cell identifier fields of the WSC transaction. If you want no acknowledgment, specify blanks. The work session controller, when it builds an acknowledgment transaction, exchanges the primary and secondary identifiers from the WSC transaction. As a result, the acknowledgment transaction is routed where you specified; you can identify it (because its secondary transaction identifier is WSC) as having been sent by the work session controller.

The format of the acknowledgment transaction is shown along with the description of each work session controller command, in chapter "Coding Work Session Controller Transactions" on page 305.

Communicating with Multiple Terminals

To use storage most efficiently, your application should communicate with multiple terminals. When you use the WSC transactions, you don't have to keep track of which terminal you're communicating with or what function is being performed for each terminal. Instead, you can design your application as if it were communicating with only one terminal.

You do need to establish a *session* with each terminal—a connection between the program and the terminal. To establish a session, you send two WSC transactions. The first is SD (start device), in which you specify the EDX terminal name and another name of your choice which you will use in all subsequent communications with that terminal. The second is SS (start session), which causes an ENQT of the terminal and, optionally, sends it a screen image. Your program can send the SD and SS transactions for each terminal with which it's going to communicate, or the terminal operator can use the >TRAN command to send them.

In every WSC transaction, you code a command to specify what you want done and a terminal's session name to specify where it's to be done. The work session controller returns that information as part of every acknowledgment transaction. You issue a RECEIVE M or RECEIVE T instruction that waits for the receipt of an acknowledgment transaction. When you receive an acknowledgment, you can examine it to determine which function was completed for which terminal and proceed accordingly.

Your application may need to know more about each terminal than its name and the function just completed for it. The work session controller allows you to save information associated with a specific terminal in storage or on disk.

When you send an SD (start device) transaction, you can request that a work area of up to 128 bytes be allocated. The work area is appended to the station block that the work session controller creates to represent the terminal. To refer to the work area, use the LOCATE instruction to get the address of the terminal's station block. The area begins at the offset defined by USERWORK, which is an equate in module S$WSCEQU. You can use the work area only when your program runs in the Series/1 to which the terminal is attached.

You can use the SV (save data) and RS (restore data) transactions to save information in a $.WSCIMG data set. You can save up to 256 records if the data set is in the same Series/1 as your program. You can save up to 8 records if the data set is in a different Series/1.

Work Session Controller Transaction Commands

The format of each work session controller transaction and the corresponding acknowledgement transactions is given in chapter "Coding Work Session Controller Transactions" on page 305. Each transaction includes a command. The commands provide these functions:

Session Control

SD	Start device; create station block to represent work session controller terminal.
SS	Start session with terminal; ENQT on terminal and, optionally, send a screen image.
ES	End session with terminal; DEQT terminal.
PD	Stop device; delete station block.
SL	Specify the station name by which the terminal is defined to the Communications Facility as an emulated 3270 device.
LI	Link to another program.

Terminal I/O Functions

BI	Send a screen image from data set $.WSCIMG to a terminal.
CD	Clear all unprotected data areas on a terminal.
RA	Read protected or unprotected data from a terminal.
RD	Read unprotected data from a terminal.
WD	Write unprotected data to a terminal or printer.
WP	Write protected data to a terminal.
PW	Write data to the priority area of a terminal.

Terminal Control

RC	Read the cursor position.
SC	Set the cursor position.
LK	Lock the keyboard.

UK	Unlock the keyboard.
LS	Set lock sequence to prevent keyboard from being unlocked when a WK command is issued.
US	End lock sequence.
TN	Sound the tone (audible alarm).
IT	Set time-out interval for subsequent WK commands.
ST	Set program function key table—a table of transaction identifiers of the transactions to be sent when an operator presses a PF key.
RT	Read program function key table.
WK	Wait for operator to press ENTER or a PF key until interval set by IT command has elapsed.
FT	Read field table associated with a screen image in data set $.WSCIMG.
CC	Skip lines or spaces on printer or roll screen terminal.
SF	Set lines per inch for printer; set output pause for roll screen terminal; set lines per page, margins, and overflow for printer or roll screen terminal.

Save and Restore Data

SV	Save data in data set $.WSCIMG.
RS	Restore data from data set $.WSCIMG.

Adding New WSC Transaction Commands

You can, if you want, create your own WSC transaction commands. To do that, you have to add an entry for your command to the table of WSC transaction commands labeled CMDTABLE in the module S$WSC.

The entry for each transaction command has this format:

```
DATA C'XX',A(routine)
```

where *XX* is the 2-character transaction command code, and *routine* is the label of the routine that is to process this request.

To add your new command, simply add an entry of this form to the table. Add your routine to S$WSC, reassemble it, and build load module $.WSC, as explained in the section "Maintaining the Communications Facility" in the *Design and Installation Guide.*

For example, say you wanted to add a transaction command "XX" which would send the message "HELLO" to a terminal. You would insert this entry into the table:

```
DATA C'XX',A(SENDH)
```

You would also add this new routine:

```
SENDH    PRINTEXT 'HELLO'
         GOTO     SD
```

where SD is the routine in the work session controller that sends a response if one was requested and frees buffers.

Storing Images in the Image Library

Images displayed through the work session controller must be in the partitioned data set $.WSCIMG, which is distributed as part of the Communications Facility. The $.WSCUT1 utility program allows you to convert images that you have created by means of the EDX $IMAGE utility to work session controller images.

Creating an Image

You use the EDX utilities $DISKUT1 and $IMAGE to create an image. Information about the EDX utilities is in the EDX *Operator Commands and Utilities Reference* manual.

Use the DISKUT1 utility to allocate a data set to store the output of $IMAGE.

Use the $IMAGE utility to build or change your image. The line length must be 80. The number of rows can be from 1 to 24. Keep the data set that contains the output of $IMAGE if you might want to modify the image in the future.

Formatting and Storing an Image

Use the SI (format and store image) command of $.WSCUT1 to convert the output of $IMAGE to a work session controller image and store it in the image library, $.WSCIMG. Note that it is not necessary to allocate a member of $.WSCIMG to hold the image; $.WSCUT1 allocates the appropriate number of records. The dialog to format and store an image is:

ENTER COMMAND (?): **SI**
ENTER $IMAGE DATA SET (NAME, VOLUME): **MENU,EDX003**
ENTER # OF FIRST IMAGE LINE (1-24): **1**
ENTER WSC IMAGE NAME: **MENU**
PRESS ENTER TO DISPLAY IMAGE
THEN PRESS ENTER AGAIN TO PROCEED

At this point, the selected image appears on the screen for verification. Press ENTER again to continue:

STORE MENU? (Y/N): **Y**
MEMBER MENU STORED

In case of a problem, you may receive one of five error messages:

ERROR *n* DETECTED DURING $IMOPEN

$.WSCUT1 uses the EDX subroutine $IMOPEN to read an image from the $IMAGE data set. This message indicates that the read failed. *n* can be:

1—Disk I/O error
2—Invalid data set name
3—Data set not found
4—Incorrect header or data set length
5—Input buffer too small
6—Invalid volume name
7—No 3101 image available
8—Data set name longer than 8 bytes

$IMAGE DATA SET INVALID FOR USE WITH WSC
  LINE SIZE NOT 80

FIRST IMAGE LINE # MUST BE BETWEEN 1 AND 24

MEMBER *name* ALREADY USED
  REPLACE? (Y/N)

NOT ENOUGH ROOM IN WSC IMAGE LIBRARY
  MEMBER *name* NOT STORED

Displaying an Image

Use the DI (display image) command to display an image in the image library, $.WSCIMG. The dialog to display an image is:

```
ENTER COMMAND (?): DI
ENTER IMAGE NAME: MENU
PRESS ENTER TO DISPLAY IMAGE
THEN PRESS ENTER AGAIN TO PROCEED
```

At this point, the selected image appears on the screen. Press ENTER again to continue.

If $.WSCUT1 doesn't find the member you request, you'll get the message "MEMBER *name* NOT FOUND".

Maintaining the Image Library

You use the EDX utility $DIUTIL to maintain data set $.WSCIMG. You can delete members, compress the data set, or copy members to a larger data set if you need to enlarge it. You can also allocate members to be used by the work session controller command SV (save data).

Using $.WSMENU

$.WSMENU is a program, distributed with the Communications Facility, that you can use to start sessions between EDX terminals and work session controller application programs.

$.WSMENU Functions

$.WSMENU displays a menu that offers the user various options. The options include several that you can use to start communication between your transaction-processing applications and their users.

The menu is distributed with the Communications Facility. It is a screen image named MENU in data set $.WSCIMG.

Running $.WSMENU

Use any of the EDX editors to insert these two transaction identifiers into data set $.SYSPD:

```
TID WSC $.WSC 42
TID MENU $.WSMENU 22
```

Start the program dispatcher:

```
> CP S $.PD
```

Issue a WSC SD (start device) transaction, as described later in this section, to start a session between a terminal and $.WSMENU. The terminal cannot be the Communications Facility system log device. You can issue the transaction from a program, or you can use the TRAN command to issue it from a terminal:

```
> TRAN transaction
```

If the work session is already started, you can also use the WSC command to generate a transaction that starts a session for the terminal where you issue the command.

In the following example transactions, the character "b" represents a blank. The transaction to start a session for EDX terminal $SYSLOGA, using the name TERM0004 for the session, is:

```
WSCbbbMENUbbSDTERM0004$SYSLOGA
```

The WSC command to generate that transaction is:

```
> WSC MENU 0 TERM0004
```

If you don't need a specific session name, you can just issue:

```
> WSC
```

and a session name that is the reverse of the EDX terminal name is generated. If you issue the command at $SYSLOGA, the session name is AGOLSYS$.

If you need to communicate with an application that requires a user work area, include the size of the work area in the transaction:

```
WSCbbbMENUbbSDTERM0004$SYSLOGA064
```

or:

```
> WSC MENU 64
```

If you plan to select the option to communicate with a host, include the terminal's station name (the name by which it is defined to the Communications Facility) after the work area field:

```
WSCbbbMENUbbSDTERM0004$SYSLOGA000T4978
```

(You can't use the WSC command in this case. It doesn't have a station-name parameter.)

The WSC transaction causes the work session controller ($.WSC) to be started if it isn't already. The transaction is sent to the work session controller, which starts the specified terminal and issues this acknowledgment transaction (where *x* is Y or N to indicate whether or not the terminal was started):

```
MENUbbWSCbbbSDTERM0004x
```

The MENU transaction causes $.WSMENU to be started if it isn't already. The transaction is sent to $.WSMENU, which starts a session with the terminal, displays the menu shown in Figure 20 on page 93 and waits for input from the terminal.

The menu offers these options:

1. Connect to a host system as a 3277 (valid only for 3101s, 4978s, and 4980s).

2. Terminate the session with this terminal and return it to EDX control.

3. Start the host management utility and remote management utility.

4. Build and send a transaction.

5-7. Send a predefined transaction called USR1, USR2, or USR3.

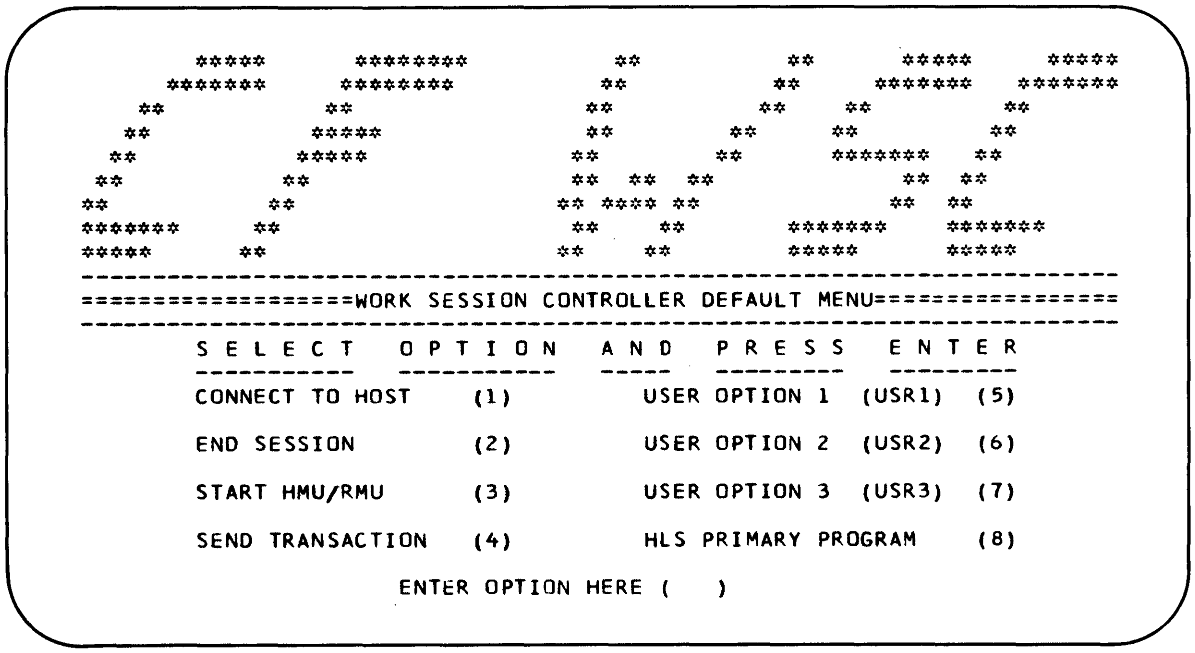
8. Connect to a program that uses the work session controller high-level language subroutines.

When you select option 1, $.WSMENU terminates the session and issues a CP S (start) command for the terminal's station name. To effect a communication to the host, that station must be linked to an appropriate emulated terminal station, and that station, plus its associated line station or SNA physical unit station, must be active.

When the you select option 2, $.WSMENU terminates the session, thus freeing the terminal for normal EDX use. $.WSMENU is ended if there are no more work session controller terminals active.

When you select option 3, $.WSMENU terminates the session and sends a transaction that loads the $.HMU program. $.HMU prompts you for a remote cell designation and sends a transaction that starts the $.RMU program. Transaction identifiers HMU and RMU must be defined to the program dispatcher. Refer to the *Operator's Guide* for more information about $.HMU and $.RMU.

When you select option 4, $.WSMENU responds with a request for data to build a transaction. Enter the entire transaction you want $.WSMENU to send.



```
        *****      ********          **          **      *****     *****
      *******     ********          **          **      *******   *******
    **           **                **          **    **          **
   **           *****              **        **     **          **
  **           *****              **        **      *******   **
 **           **                  **  **  **             **  **
**           **                  ** **** **             **  **
*******     **                    **    **       *******    *******
*****      **                    **    **        *****      *****
------------------------------------------------------------------------
===================WORK SESSION CONTROLLER DEFAULT MENU=================
------------------------------------------------------------------------
        S E L E C T   O P T I O N   A N D   P R E S S   E N T E R
        -----------   -----------   -----   ---------   ---------
        CONNECT TO HOST      (1)         USER OPTION 1   (USR1)   (5)

        END SESSION          (2)         USER OPTION 2   (USR2)   (6)

        START HMU/RMU        (3)         USER OPTION 3   (USR3)   (7)

        SEND TRANSACTION     (4)         HLS PRIMARY PROGRAM      (8)

                       ENTER OPTION HERE (     )
```

Figure 20. $.WSMENU User Menu

When you select one of options 5-7, $.WSMENU sends a PD (stop device) transaction, thus freeing the terminal for use by an application program. It then builds and sends a transaction with this format:

1-4	5-6	7-12	13-14	15-22	23-30	31-33	34-41
USR*x*	*pc*	MENU??	SD	*terminal*	*EDXname*	*size*	*staname*

USR*x*
: is the primary transaction identifier, where *x* is 1, 2, or 3.

pc
: is the primary cell identifier (the cell in which $.WSMENU is running).

MENU??
: is the secondary transaction identifier and cell identifier.

SD
: is the start device work session controller command.

terminal
: is the terminal name used for this session.

EDXname
: is the EDX terminal name.

size
is the size of the user work area at the end of the station block for the terminal.

staname
is the terminal's station name.

Transaction identifier USR*x* must be defined to the program dispatcher. The transaction causes a program (the one associated with USR*x*) to be started if it isn't already, and the transaction is sent to the program. If the program is to communicate with the terminal through the work session controller, it just sends the transaction after first changing the primary transaction identifier to WSC and the secondary transaction identifier and cell identifier to indicate where the acknowledgment is to be sent.

When you select option 8, $.WSMENU displays the high-level language subroutines primary menu, from which you can select an application program that uses the subroutines.

Example Transaction-Processing Application

The sample application in this section illustrates program segmentation, techniques for coding various types of transaction-processing programs, and the use of work session controller transactions.

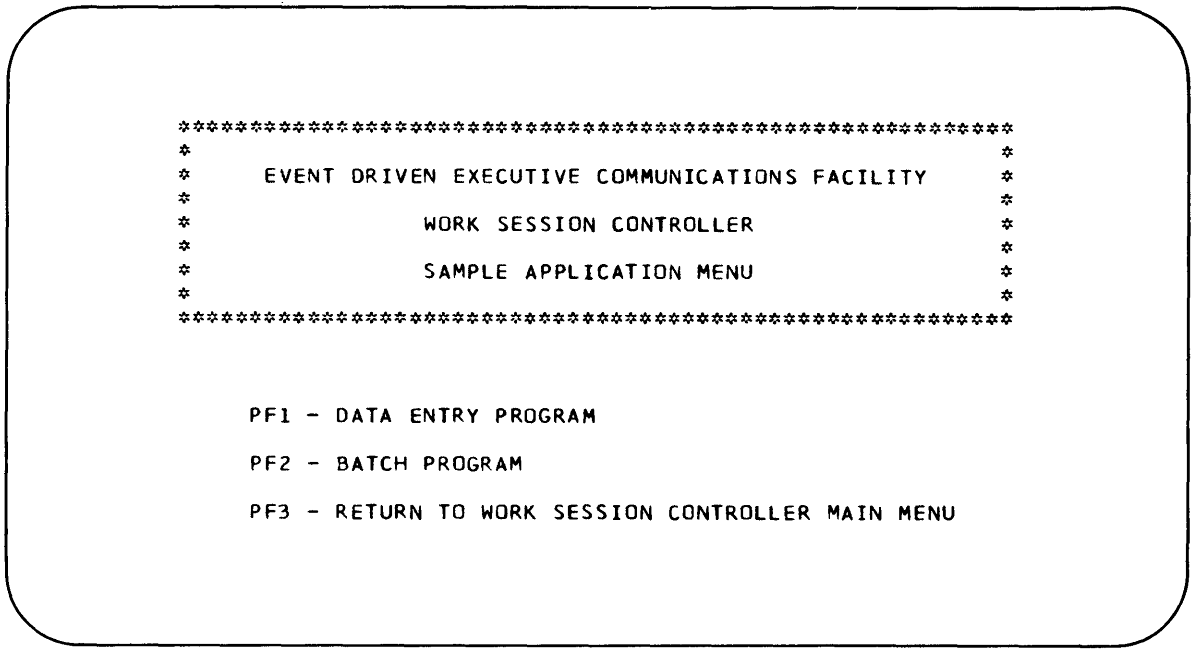
The application is designed to be activated by option 5 of $.WSMENU. Option 5 causes transaction USR1 to be issued, as discussed under "Running $.WSMENU" on page 91. Once the terminal user has selected option 5, the application menu shown in Figure 21 on page 95 is displayed.

The user can choose to:

- Call a data entry program, which displays the data entry screen shown in Figure 22 on page 96. The data entered on this screen is written to a disk file. When the user has no more data to enter, pressing PF3 causes a return to the application menu.
- Call a batch program, which makes a backup copy of the data entry file.
- Return to the main menu (the one displayed by $.WSMENU).

The application consists of four programs. The interaction between them and $.WSMENU requires that these transaction identifiers be defined to the program dispatcher:

```
TID WSC $.WSC 42
TID MENU $.WSMENU 22
TID USR1 STRTSESS 22
TID MEN1 MENUPROG 12
TID ENT1 ENTRPROG 32,P
TID BCH1 BTCHPROG 20
```



```
**********************************************************
*                                                        *
*    EVENT DRIVEN EXECUTIVE COMMUNICATIONS FACILITY      *
*                                                        *
*               WORK SESSION CONTROLLER                  *
*                                                        *
*               SAMPLE APPLICATION MENU                  *
*                                                        *
**********************************************************


     PF1 - DATA ENTRY PROGRAM

     PF2 - BATCH PROGRAM

     PF3 - RETURN TO WORK SESSION CONTROLLER MAIN MENU
```

Figure 21. Sample Transaction-Processing Program Menu

STRTSESS Program

Program STRTSESS is started when the user selects option 5 from the $.WSMENU menu. It issues three work session controller transactions:

- SD (start device), using the data passed in the USR1 transaction
- SS (start session)
- LI (link), with the secondary transaction identifier MEN1.

STRTSESS is a type 22 program. It receives multiple transactions and terminates when its work is finished—in this example, when its message queue is empty. Thus it may start just one session, or it may start multiple sessions if several users log on at approximately the same time. Figure 23 on page 98 is a listing of STRTSESS.

MENUPROG Program

Program MENUPROG is started when the work session controller issues the acknowledgment of STRTSESS's link transaction. It terminates after issuing three work session controller transactions:

- BI (build image), to display the application menu
- ST (set program function key table)
- WK (wait for key)

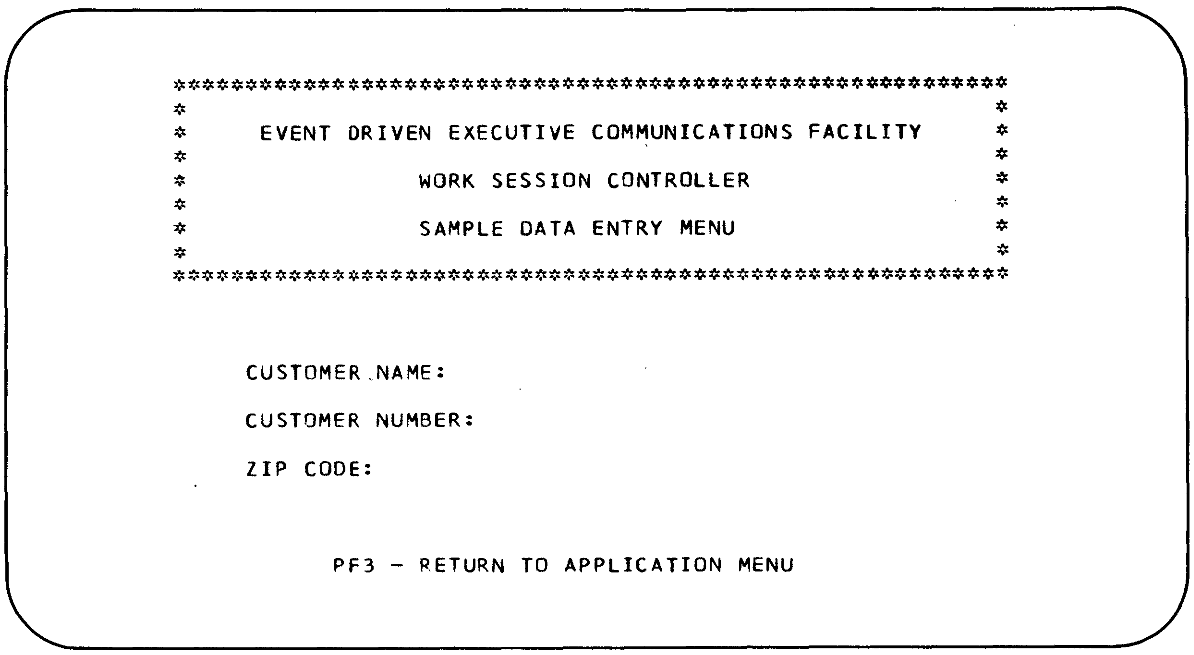

```
*************************************************************
*                                                           *
*     EVENT DRIVEN EXECUTIVE COMMUNICATIONS FACILITY        *
*                                                           *
*                 WORK SESSION CONTROLLER                   *
*                                                           *
*                 SAMPLE DATA ENTRY MENU                    *
*                                                           *
*************************************************************


     CUSTOMER NAME:

     CUSTOMER NUMBER:

     ZIP CODE:



           PF3 - RETURN TO APPLICATION MENU
```

Figure 22. Sample Data Entry Screen

The ST and WK transactions establish the following associations between PF keys and transaction identifiers:

PF1—ENT1	PF2—MEN1
PF3—MEN1	other—MEN1

Thus MENUPROG is also started when the user presses ENTER or a PF key other than PF1 and the work session controller issues the acknowledgment of the WK transaction. The program proceeds according to the interrupt key code in the WK acknowledgment.

If the user presses PF2, the program issues a WP (write protected) transaction to notify the user that the batch job has been initiated. The secondary transaction identifier is BCH1. The program then issues a WK transaction and terminates. The user may again select an option from the application menu.

If the user presses PF3, MENUPROG issues an ES (end session) transaction with the secondary transaction identifier MENU, and terminates. The acknowledgment of the SS transaction is sent to $.WSMENU, which displays the main menu.

If the user presses ENTER or any other PF key, MENUPROG proceeds as it does when it receives the acknowledgment of STARTSESS's link transaction.

MENUPROG is a type 12 program. It receives one transaction (either an LI or a WK acknowledgment), processes it, and terminates. Note that MENUPROG routes the WK acknowledgment to itself. For other transactions, MENUPROG specifies the secondary transaction identifier as BCH1, MENU, or blanks (no acknowledgment). Figure 24 on page 100 is a listing of MENUPROG.

ENTRPROG Program

Program ENTRPROG is started the first time a user presses PF1 from the application menu and the work session controller issues the acknowledgment of MENUPROG's WK transaction. Each time ENTRPROG receives such an acknowledgment, it issues transactions to display the data entry screen, reset the program function key stack, and wait for a key.

If the user presses PF3, the program issues a link transaction with the secondary transaction identifier MEN1, which causes a return to the application menu. Otherwise, the program reads the data from the screen, writes it to a file (CUSTFILE), clears the user's input from the screen, and waits for a key.

Because ENTRPROG is designed to communicate with multiple users, it has to keep track of the next operation to be performed for each user. It does so by storing a "next operation" code in the user work area in the terminal station block.

ENTRPROG is a type 32 program. It receives multiple transactions, and terminates when it receives a stop request in a status message. A stop request may be issued because the application is being halted or (since the definition of ENT1 specified that the program is purgable) because the program dispatcher needs storage for another program. Before terminating, ENTRPROG processes any pending transactions on its message queue.

Note that ENTRPROG routes all transaction acknowledgments, other than the link to MENUPROG, to itself. As a result, if the program is purged but the application is not halted, the next acknowledgment causes the program to be restarted and it continues where it left off. Figure 25 on page 102 is a listing of ENTRPROG.

BTCHPROG Program

Program BTCHPROG is started when the work session controller issues the acknowledgment of MENUPROG's WP transaction. It copies the data entry file (CUSTFILE) to another file (BTCHFILE) and terminates.

BTCHPROG is a type 20 program. As long as it is running, no second copy of it is loaded. Multiple copies are not allowed because all users of the application share the files. Figure 26 on page 105 is a listing of BTCHPROG.

```
STRTSESS PROGRAM  START
*
***********************************************************************
* FUNCTION:          START SESSION WITH TERMINAL USER                 *
* TRANSACTION ID:    USR1                                             *
* PROGRAM TYPE:      22                                               *
***********************************************************************
*
START    EQU   *
         RECEIVE M,WSCTRAN,EXIT=ERR01          GET TRANSACTION
         CALL  GETACK                          EXTRACT COMMON FIELDS
         IF    (WSCCMD,EQ,SDCMD),GOTO,SD       FROM $.WSMENU
         IF    (WSCCMD,EQ,SSCMD),GOTO,LI       SESSION STARTED
         GOTO  START                           IGNORE ANY OTHER MESSAGE
*
***********************************************************************
* PROCESS USR1 TRANSACTION SENT BY $.WSMENU:                          *
*   USR1  WSC   SDTERMNAMEDEVNAME 000STANAME                          *
***********************************************************************
*
SD       EQU   *
         GET   F,DEVNAME,WSCTRAN               GET EDX DEVICE NAME
         GET   F,WORK,WSCTRAN                  IGNORE WORK AREA SIZE
         GET   F,STNAME,WSCTRAN                GET CF STATION NAME
         CALL  PUTINIT,(BLKTID),(SDCMD)        BUILD SD TRANSACTION,
*                                              ... NO SECONDARY TID
         PUT   F,WSCTRAN,DEVNAME               APPEND EDX DEVICE NAME
         PUT   F,WSCTRAN,WORKSPAC              APPEND WORK AREA SIZE
         PUT   F,WSCTRAN,STNAME                APPEND CF STATION NAME
         SEND  MT,,WSCTRAN,EXIT=ERR01          SEND SD TRANSACTION
         CALL  PUTINIT,(PRITID),(SSCMD)        BUILD SS TRANSACTION,
*                                              ... SECONDARY TID = USR1
         PUT   F,WSCTRAN,'Y'                   WAIT FOR TERMINAL
         SEND  MT,,WSCTRAN,EXIT=ERR01          SEND SS TRANSACTION
         GOTO  START                           WAIT FOR ACKNOWLEDGMENT
*
***********************************************************************
* PROCESS ACKNOWLEDGEMENT OF SS TRANSACTION.                          *
***********************************************************************
*
LI       EQU   *
         GET   F,CODE,WSCTRAN                  CHECK SS RETURN CODE
         IF    (CODE,EQ,C'N',BYTE),GOTO,ERR02
         CALL  PUTINIT,(MENUTID),(LICMD)       BUILD LI TRANSACTION,
*                                              ... SECONDARY TID = MEN1
         SEND  MT,,WSCTRAN,EXIT=ERR01          SEND TRANSACTION
ENDPROG  EQU   *
         RECEIVE NOTIFY,WAIT=NO                ANY MORE MESSAGES?
         IF    (STRTSESS,LT,0),GOTO,START      YES, CONTINUE
         LOCATE  ST,#1,OPTION=PROGSTOP         PURGE STATION BLOCK & STOP
*
```

Figure 23 (Part 1 of 2). STRTSESS Listing

```
***********************************************************************
* GETACK: SUBROUTINE TO EXTRACT COMMON FIELDS FROM TRANSACTION.       *
***********************************************************************
*
         SUBROUT GETACK
         GET   F,PRITID,WSCTRAN       PRIMARY TID/CELL (USR1..)
         GET   F,SECTID,WSCTRAN       SECONDARY TID/CELL (WSC ..)
         GET   F,WSCCMD,WSCTRAN       WSC COMMAND
         GET   F,TERMNAME,WSCTRAN     TERMINAL NAME
         RETURN
*
***********************************************************************
* PUTINIT: SUBROUTINE TO BUILD COMMON FIELDS OF TRANSACTION.          *
*          PARM-1 = SECONDARY TID/CELL                                *
*          PARM-2 = WSC COMMAND                                       *
***********************************************************************
*
         SUBROUT PUTINIT,PUTTID,PUTCMD
         PUT   F,WSCTRAN,WSCTID,OPTION=INITIAL  PRI TID/CELL (WSC ..)
         PUT   F,WSCTRAN,PUTTID*      SECONDARY TID/CELL AS SPECIFIED
         PUT   F,WSCTRAN,PUTCMD*      WSC COMMAND AS SPECIFIED
         PUT   F,WSCTRAN,TERMNAME     TERMINAL NAME
         RETURN
*
***********************************************************************
* ERROR ROUTINES -- LOG MESSAGE AND TERMINATE.                        *
***********************************************************************
*
ERR01    EQU   *                      SEND/RECEIVE ERROR
         SEND  E,99,'SEND/RECEIVE ERROR',XCODE=STRTSESS*
         GOTO  ENDPROG
ERR02    EQU   *                      SS COMMAND FAILED
         SEND  E,98,'CANNOT START SESSION'
         GOTO  ENDPROG
*
***********************************************************************
* DATA AREAS                                                          *
***********************************************************************
*
WSCTRAN  DEFINE  BUFFER,SIZE=80   WSC TRANSACTION BUFFER
WSCTID   TEXT  'WSC   '           WORK SESSION CONTROLLER TID/CELL
PRITID   TEXT  LENGTH=6           PRIMARY TID/CELL
SECTID   TEXT  LENGTH=6           SECONDARY TID/CELL
WSCCMD   TEXT  LENGTH=2           WSC COMMAND
TERMNAME TEXT  LENGTH=8           TERMINAL NAME
CODE     TEXT  LENGTH=1           RETURN CODE FROM SS COMMAND
DEVNAME  TEXT  LENGTH=8           EDX DEVICE NAME
WORK     TEXT  LENGTH=3           WORKSPACE FROM $.WSMENU - IGNORE
STNAME   TEXT  LENGTH=8           CF STATION NAME
WORKSPAC TEXT  '010'              WORK AREA SIZE FOR APPLICATION
MENUTID  TEXT  'MEN1??'           TID/CELL FOR PROGRAM 'MENUPROG'
BLKTID   TEXT  LENGTH=6           BLANK TID/CELL = NO ACK
SDCMD    TEXT  'SD'               START DEVICE COMMAND
SSCMD    TEXT  'SS'               START SESSION COMMAND
LICMD    TEXT  'LI'               LINK COMMAND
         ENDPROG
         END
```

Figure 23 (Part 2 of 2). STRTSESS Listing

```
MENUPROG PROGRAM START
*
***********************************************************************
* FUNCTION:         DISPLAY APPLICATION MENU OR PROCESS USER'S INPUT  *
* TRANSACTION ID:   MEN1                                              *
* PROGRAM TYPE:     12                                                *
***********************************************************************
*
START    EQU   *
         RECEIVE  M,WSCTRAN,EXIT=ERR01        GET TRANSACTION
         CALL   GETACK                        EXTRACT COMMON FIELDS
*
***********************************************************************
* PROCESS ACKNOWLEDGMENT OF WK TRANSACTION.                           *
***********************************************************************
*
         IF     (WSCCMD,EQ,WKCMD)             WSC COMMAND WAS WK
           GET    F,ROWCOL,WSCTRAN            IGNORE CURSOR ROW/COLUMN
           GET    F,KEYCODE,WSCTRAN           CHECK KEY CODE
           IF     (KEYCODE,EQ,PF2,3)          PF2 = EXECUTE BATCH JOB
             CALL PUTINIT,(BCHTID),(WPCMD)    BUILD WP TRANSACTION,
*                                             ... SECONDARY TID = BCH1
             PUT  F,WSCTRAN,ROWCOL1           CURSOR AT ROW 23, COL 10
             PUT  F,WSCTRAN,MESSAGE           MESSAGE FOR USER
             SEND MT,,WSCTRAN,EXIT=ERR01      SEND TRANSACTION
             GOTO WK                          DO WAIT KEY TRANSACTION
           ENDIF
           IF   (KEYCODE,EQ,PF3,3)            PF3 = RETURN TO $.WSMENU
             CALL PUTINIT,(BLKTID),(ESCMD)    BUILD ES TRANSACTION,
*                                             ... NO SECONDARY TID
             SEND MT,,WSCTRAN,EXIT=ERR01      SEND TRANSACTION
             CALL PUTINIT,(WSCMTID),(SSCMD)   BUILD SS TRANSACTION,
*                                             ... SECONDARY TID = MENU
             PUT  F,WSCTRAN,'Y'               WAIT FOR TERMINAL
             SEND MT,,WSCTRAN,EXIT=ERR01      SEND TRANSACTION
             GOTO ENDPROG                     DONE PROCESSING
           ENDIF
         ENDIF
*
***********************************************************************
* PROCESS ACKNOWLEDGEMENT OF LI TRANSACTION ISSUED BY STRTSESS        *
* OR USER INPUT OTHER THAN PF2 OR PF3.                                *
***********************************************************************
*
         CALL   PUTINIT,(BLKTID),(BICMD)      BUILD BI TRANSACTION,
*                                             ... NO SECONDARY TID
         PUT    F,WSCTRAN,MENUIMAG            APPEND IMAGE NAME
         SEND   MT,,WSCTRAN,EXIT=ERR01        SEND TRANSACTION
         CALL   PUTINIT,(BLKTID),(STCMD)      BUILD ST TRANSACTION,
*                                             ... NO SECONDARY TID
         PUT    F,WSCTRAN,PFKSTACK            APPEND PF KEY SETTINGS
         SEND   MT,,WSCTRAN,EXIT=ERR01        SEND TRANSACTION
WK       EQU    *
         CALL   PUTINIT,(MENUTID),(WKCMD)     BUILD WK TRANSACTION,
*                                             ... SECONDARY TID = MEN1
         SEND   MT,,WSCTRAN,EXIT=ERR01        SEND TRANSACTION
ENDPROG  EQU    *
         LOCATE   ST,#1,OPTION=DELETE         DELETE STATION IF NO MSGS
         PROGSTOP   LOGMSG=NO                 STOP
*
```

Figure 24 (Part 1 of 2). MENUPROG Listing

```
************************************************************************
* GETACK: SUBROUTINE TO EXTRACT COMMON FIELDS FROM TRANSACTION.        *
************************************************************************
*
         SUBROUT GETACK
         GET    F,PRITID,WSCTRAN         PRIMARY TID/CELL (USR1..)
         GET    F,SECTID,WSCTRAN         SECONDARY TID/CELL (WSC ..)
         GET    F,WSCCMD,WSCTRAN         WSC COMMAND
         GET    F,TERMNAME,WSCTRAN       TERMINAL NAME
         RETURN
*
************************************************************************
* PUTINIT: SUBROUTINE TO BUILD COMMON FIELDS OF TRANSACTION.           *
*           PARM-1 = SECONDARY TID/CELL                                *
*           PARM-2 = WSC COMMAND                                       *
************************************************************************
*
         SUBROUT PUTINIT,PUTTID,PUTCMD
         PUT    F,WSCTRAN,WSCTID,OPTION=INITIAL  PRI TID/CELL (WSC ..)
         PUT    F,WSCTRAN,PUTTID*        SECONDARY TID/CELL AS SPECIFIED
         PUT    F,WSCTRAN,PUTCMD*        WSC COMMAND AS SPECIFIED
         PUT    F,WSCTRAN,TERMNAME       TERMINAL NAME
         RETURN
*
************************************************************************
* ERROR ROUTINE -- LOG MESSAGE AND TERMINATE.                          *
************************************************************************
*
ERRC1    EQU    *
         SEND   E,99,'SEND/RECEIVE ERROR',XCODE=MENUPROG*
         GOTO   ENDPROG
*
************************************************************************
* DATA AREAS                                                           *
************************************************************************
*
WSCTRAN  DEFINE BUFFER,SIZE=80       WSC TRANSACTION BUFFER
WSCTID   TEXT   'WSC   '             WORK SESSION CONTROLLER TID/CELL
PRITID   TEXT   LENGTH=6             PRIMARY TID/CELL
SECTID   TEXT   LENGTH=6             SECONDARY TID/CELL
WSCCMD   TEXT   LENGTH=2             WSC COMMAND
TERMNAME TEXT   LENGTH=8             TERMINAL NAME
PFKSTACK TEXT   'ENT1  MEN1  MEN1    ',LENGTH=30   PF1 - PF3 SETTINGS
KEYCODE  TEXT   LENGTH=3             KEY CODE FROM WK ACKNOWLEDGMENT
PF2      TEXT   '  2'                PF 2 KEY CODE
PF3      TEXT   '  3'                PF 3 KEY CODE
MENUIMAG TEXT   'MENUIMAG'           APPLICATION MENU NAME
WSCMTID  TEXT   'MENU??'             TID/CELL FOR $.WSMENU
MENUTID  TEXT   'MEN1??'             TID/CELL FOR MENUPROG
BCHTID   TEXT   'BCH1??'             TID/CELL FOR BTCHPROG
BLKTID   TEXT   LENGTH=6             BLANK TID/CELL = NO ACK
ROWCOL   TEXT   LENGTH=4             ROW & COL FROM WK - IGNORE
ROWCOL1  TEXT   '2310'               ROW & COL FOR WP TRANSACTION
BICMD    TEXT   'BI'                 BUILD IMAGE COMMAND
STCMD    TEXT   'ST'                 SET PFKEY STACK COMMAND
WKCMD    TEXT   'WK'                 WAIT KEY COMMAND
ESCMD    TEXT   'ES'                 END SESSION COMMAND
SSCMD    TEXT   'SS'                 START SESSION COMMAND
WPCMD    TEXT   'WP'                 WRITE PROTECTED COMMAND
MESSAGE  TEXT   '*** BATCH JOB INITIATED ***'
         ENDPROG
         END
```

Figure 24 (Part 2 of 2). MENUPROG Listing

```
ENTRPROG PROGRAM   START,DS=((CUSTFILE,EDX002))
*
***********************************************************************
* FUNCTION:          DISPLAY DATA ENTRY SCREEN AND PROCESS USER INPUT   *
* TRANSACTION ID:    ENT1                                               *
* PROGRAM TYPE:      32                                                 *
***********************************************************************
*
         COPY   S$CFEQU
         COPY   S$WSCEQU
START    EQU    *
         IF     (QUIT,EQ,1)                   STOP WAS REQUESTED
           RECEIVE  NOTIFY,WAIT=NO            ANY PENDING MESSAGES?
           IF   (ENTRPROG,GT,0),GOTO,ENDPROG  NO, STOP
         ENDIF
         RECEIVE  M,WSCTRAN                   GET TRANSACTION
         IF     (ENTRPROG,GT,0)               ERROR?
           IF     (ENTRPROG,NE,6),GOTO,ERR01  YES
*                                             NO - STATUS MESSAGE
           MOVE   QUIT,1                      SET STOP FLAG
           GOTO   START                       PROCESS PENDING MSGS
         ENDIF
         CALL   GETACK                        EXTRACT COMMON FIELDS
*
***********************************************************************
* GET ADDRESS OF TERMINAL'S STATION BLOCK AND PROCEED ACCORDING TO     *
* 'NEXT OPERATION' CODE IN WORK AREA.                                  *
***********************************************************************
*
         LOCATE   ST,#1,TERMNAME,EXIT=ERR01
         GOTO   (BI,ST,SC,WK,CHKKEY,RD,SC),(USERWORK,#1)
*
BI       EQU    *       CODE = 0, DISPLAY DATA ENTRY SCREEN
         CALL   PUTINIT,(PRITID),(BICMD)      BUILD BI TRANSACTION,
*                                             ... SECONDARY TID = ENT1
         PUT    F,WSCTRAN,ENTRIMAG            APPEND IMAGE NAME
         CALL   SENDTRAN,1                    SEND AND SET CODE = 1
         GOTO   START                         GET ACKNOWLEDGEMENT
*
ST       EQU    *       CODE = 1, RESET PF KEY STACK
         CALL   PUTINIT,(PRITID),(STCMD)      BUILD ST TRANSACTION,
*                                             ... SECONDARY TID = ENT1
         PUT    F,WSCTRAN,PFKSTACK            APPEND PF KEY SETTINGS
         CALL   SENDTRAN,2                    SEND AND SET CODE = 2
         GOTO   START                         GET ACKNOWLEDGMENT
*
SC       EQU    *       CODE = 2 OR 6, SET CURSOR TO INPUT AREA
         CALL   PUTINIT,(PRITID),(SCCMD)      BUILD SC TRANSACTION,
*                                             ... SECONDARY TID = ENT1
         PUT    F,WSCTRAN,ROWCOL1             APPEND ROW/COLUMN
         CALL   SENDTRAN,3                    SEND AND SET CODE = 3
         GOTO   START                         GET ACKNOWLEDGEMENT
*
WK       EQU    *       CODE = 3, SEND WAIT KEY TRANSACTION
         CALL   PUTINIT,(PRITID),(WKCMD)      BUILD WK TRANSACTION,
*                                             ... SECONDARY TID = ENT1
         CALL   SENDTRAN,4                    SEND AND SET CODE = 4
         GOTO   START                         GET ACKNOWLEDGEMENT
```

Figure 25 (Part 1 of 3). ENTRPROG Listing

```
*
CHKKEY   EQU   *        CODE = 4, GET KEY CODE FROM WK ACKNOWLEDGEMENT
         GET   F,ROWCOL,WSCTRAN              IGNORE CURSOR ROW/COLUMN
         GET   F,KEYCODE,WSCTRAN             CHECK KEY CODE
         IF    (KEYCODE,EQ,PF3,3)            PF3 = RETURN TO MENUPROG
           CALL   PUTINIT,(MENUTID),(LICMD)  BUILD LI TRANSACTION,
*                                            ... SECONDARY TID = MEN1
           CALL   SENDTRAN,0                 SEND AND SET CODE = 0
           GOTO   START                      GET ACK FROM ANOTHER USER
*                                            ... OR WHEN THIS ONE
*                                            ... SELECTS MENU OPTION 1
         ELSE
           CALL   PUTINIT,(PRITID),(RDCMD)   BUILD RD TRANSACTION,
*                                            ... SECONDARY TID = ENT1
           PUT    F,WSCTRAN,ROWCOL1          SET INPUT AREA ROW/COLUMN
           PUT    F,WSCTRAN,COUNTIN          SET MAXIMUM DATA LENGTH
           CALL   SENDTRAN,5                 SEND AND SET CODE = 5
           GOTO   START                      GET ACKNOWLEDGMENT
         ENDIF
*
RD       EQU   *        CODE = 5, PROCESS USER INPUT (RD ACKNOWLEDGMENT)
*                                 CLEAR USER INPUT FROM SCREEN
         GET   F,ROWCOL,WSCTRAN              IGNORE CURSOR ROW/COLUMN
         GET   F,DATACNT,WSCTRAN             GET DATA LENGTH
         GET   F,CUSTDATA,WSCTRAN            GET DATA
         WRITE DS1,CUSTDATA-2,1,ERROR=ERRO2  WRITE DATA TO CUSTFILE
         CALL  PUTINIT,(PRITID),(CDCMD)      BUILD CD TRANSACTION,
*                                            ... SECONDARY TID = ENT1
         CALL  SENDTRAN,6                    SEND AND SET CODE = 6
         GOTO  START                         GET ACKNOWLEDGMENT
*
**********************************************************************
* HAVE BEEN ASKED TO STOP AND HAVE NO MORE STORAGE-QUEUED MESSAGES.  *
**********************************************************************
*
ENDPROG  EQU   *
         LOCATE   ST,#1,OPTION=PROGSTOP      PURGE STATION BLOCK & STOP
*
**********************************************************************
* GETACK: SUBROUTINE TO EXTRACT COMMON FIELDS FROM TRANSACTION.      *
**********************************************************************
*
         SUBROUT GETACK
         GET    F,PRITID,WSCTRAN       PRIMARY TID/CELL (USR1..)
         GET    F,SECTID,WSCTRAN       SECONDARY TID/CELL (WSC ..)
         GET    F,WSCCMD,WSCTRAN       WSC COMMAND
         GET    F,TERMNAME,WSCTRAN     TERMINAL NAME
         RETURN
*
**********************************************************************
* PUTINIT: SUBROUTINE TO BUILD COMMON FIELDS OF TRANSACTION.         *
*          PARM-1 = SECONDARY TID/CELL                               *
*          PARM-2 = WSC COMMAND                                      *
**********************************************************************
*
         SUBROUT PUTINIT,PUTTID,PUTCMD
         PUT    F,WSCTRAN,WSCTID,OPTION=INITIAL  PRI TID/CELL (WSC ..)
         PUT    F,WSCTRAN,PUTTID*      SECONDARY TID/CELL AS SPECIFIED
         PUT    F,WSCTRAN,PUTCMD*      WSC COMMAND AS SPECIFIED
         PUT    F,WSCTRAN,TERMNAME     TERMINAL NAME
         RETURN
*
```

Figure 25 (Part 2 of 3). ENTRPROG Listing

```
***********************************************************************
* SENDTRAN: SUBROUTINE TO SAVE 'NEXT OPERATION' CODE IN WORK AREA IN  *
*           TERMINAL'S STATION BLOCK AND SEND THE TRANSACTION IN      *
*           BUFFER WSCTRAN.                                           *
***********************************************************************
*
         SUBROUT SENDTRAN,SNXTCMD
         MOVE  (USERWORK,#1),SNXTCMD   MOVE SPECIFIED CODE TO WORK AREA
         SEND  MT,,WSCTRAN,EXIT=ERR01  SEND TRANSACTION
         RETURN
*
***********************************************************************
* ERROR ROUTINES -- LOG MESSAGE AND TERMINATE.                        *




***********************************************************************
*
ERR01    EQU   *
         SEND  E,99,'SEND/RECEIVE ERROR',XCODE=ENTRPROG*
         GOTO  ENDPROG
ERR02    EQU   *
         SEND  E,97,'DISK I/O ERROR',XCODE=ENTRPROG*
         GOTO  ENDPROG
*
***********************************************************************
* DATA AREAS                                                          *
***********************************************************************
*
WSCTRAN  DEFINE BUFFER,SIZE=200    WSC TRANSACTION BUFFER
WSCTID   TEXT  'WSC   '            WORK SESSION CONTROLLER TID/CELL
PRITID   TEXT  LENGTH=6            PRIMARY TID/CELL
SECTID   TEXT  LENGTH=6            SECONDARY TID/CELL
WSCCMD   TEXT  LENGTH=2            WSC COMMAND
TERMNAME TEXT  LENGTH=8            TERMINAL NAME
PFKSTACK TEXT  LENGTH=30           RESET PF KEY STACK TO BLANKS
ENTRIMAG TEXT  'ENTRIMAG'          NAME OF DATA ENTRY SCREEN IMAGE
MENUTID  TEXT  'MEN1??'            TID/CELL FOR MENUPROG
BICMD    TEXT  'BI'                BUILD IMAGE COMMAND
STCMD    TEXT  'ST'                SET PF KEY STACK COMMAND
SCCMD    TEXT  'SC'                SET CURSOR COMMAND
WKCMD    TEXT  'WK'                WAIT KEY COMMAND
RDCMD    TEXT  'RD'                READ DATA COMMAND
LICMD    TEXT  'LI'                LINK COMMAND
CDCMD    TEXT  'CD'                CLEAR DATA COMMAND
ROWCOL   TEXT  LENGTH=4            ROW & COL FROM WK ACK - IGNORE
ROWCOL1  TEXT  '1427'              ROW & COL FOR SET CURSOR TRANSACTION
DATACNT  TEXT  LENGTH=4            NUMBER CHARACTERS READ
KEYCODE  TEXT  LENGTH=3            KEY OPERATOR PRESSED
COUNTIN  TEXT  '0100'
CUSTDATA TEXT  LENGTH=254          DATA RECORD WRITTEN TO DISK
PF3      TEXT  '  3'
QUIT     DATA  F'0'                FLAG TO PURGE SELF
         ENDPROG
         END
```

Figure 25 (Part 3 of 3). ENTRPROG Listing

```
BTCHPROG PROGRAM   START,DS=((CUSTFILE,EDX002),(BTCHFILE,EDX002))
*
***********************************************************************
* FUNCTION:          MAKE BACKUP COPY OF DATA ENTRY FILE             *
* TRANSACTION ID:    BCH1                                            *
* PROGRAM TYPE:      20                                              *
***********************************************************************
*
START    EQU      *
         READ     DS1,BUFFER,1,ERROR=ENDPROG      READ CUSTFILE
         WRITE    DS2,BUFFER,1,ERROR=ENDPROG      WRITE BTCHFILE
         GOTO     START
ENDPROG  PROGSTOP LOGMSG=NO                       STOP
BUFFER   BUFFER   256,BYTES
         ENDPROG
         END
```

Figure 26. BTCHPROG Listing

Writing an X.25 Application Program

An X.25 application program is a program that communicates with another X.25 program or device through a circuit station. Circuit stations are used only to route messages from one Series/1 to another over an X.25 connection.

When you write an X.25 application program, you need to:

- Decide which kind of circuit station you'll need to use
- Know how to establish a call; that is, a connection with another Data Terminal Equipment (DTE)
- Understand the format of the messages you'll send and receive.

X.25-defined packets other than data packets (for example, reset request packets, incoming call packets, etc.) are referred to in this chapter as *X.25 control packets.* Within the Communications Facility node, the corresponding information is recorded in fixed format messages referred to as *X.25 control messages.* Refer to the CCITT Recommendation X.25 if you need more detailed information on the procedures and terms used in this chapter.

Determining the Circuit Usage

If you need only to send and receive data, use a circuit station with usage STD. When you do so, you don't have to establish the call; the X.25 I/O control program establishes the call when the circuit station is started. The X.25 I/O control program handles any control packets that come across the circuit.

If you need to control the state of the X.25 circuit or get information about network conditions, use a circuit station with usage STD+. When you do so, you have to establish the call (if the circuit is a switched virtual circuit) by sending and receiving X.25 control messages. Once the connection is made, you can send and receive data and control messages.

Managing the Circuit Station

The circuit station you communicate with is managed by the X.25 IOCP. However, no productive communication will occur until its link (your program) is active. You should start the user station that represents your program before you start the circuit station. If your program is the only one that ever communicates with that circuit, you can stop the circuit station before terminating your program, as illustrated in the following example. The example depends on the user station's being directly linked to the circuit station.

```
STOPMSG  TEXT    C'P XXXXXXXX'
STOPSTA  EQU     STOPMSG+2
           •
           •
         LOCATE  ST,#1                              FIND YOUR STATION
         LOCATE  NA,#1,(Q$DLV,#1),EXIT=NOLINK       FIND ITS LINK
         MOVE    STOPSTA,(Q$NAME,#1),(8,BYTES)      GET CIRCUIT NAME
         SEND    CP,STOPMSG,ACK=YES                 SEND STOP COMMAND
           •
           •
```

When the IOCP detects that the circuit station has been stopped, it clears the call if one is still established.

Using X.25 Headers

Each message that you send or receive (either data or control) begins with a header that is referred to in this chapter as the *X.25 header*. This is not a Communications Facility message header or an X.25 packet header. It is a header built and used by the X.25 IOCP. The IOCP appends the header to messages that come across the circuit before sending them on to their destination. It removes the header from messages before sending them across the circuit.

The format of the X.25 header is:

Byte 1:	Length of the header, including the first byte
Byte 2:	Bits 0-1: Format identifier
	Bits 2-7: Message type code

With the present version of X.25 support, the X.25 header length is always 2 bytes and the format identifier is always 00. You should not assume either of these in your program. X.25 support in the future could include headers of different lengths or with different identifiers.

When you receive a message, take the header length from the header itself. You don't have to worry about the format identifier in data messages, but you do in control messages. A format identifier other than 00 may mean that the control message is not in the format described in this chapter; you should terminate your program on this condition.

The message type code for data messages is 000000. The message type codes for control messages are given in the section "X.25 Control Messages" on page 116.

Communicating with an STD Usage Circuit

To communicate with an STD usage circuit, set a direct link between your program's user station and the circuit station:

```
> CP LINK program circuit BOTH
```

You don't have to specify a destination when you send messages; they'll be sent to the circuit station. All data messages that come across the circuit will be put on your station's message queue.

Establishing the Call

You don't need to do anything in your program to establish the call. If the circuit is a permanent virtual circuit (PVC), and the circuit station is started, the connection is always established. If the circuit is a switched virtual circuit (SVC), the call is either initiated automatically (contact type INIT) or accepted automatically (contact type WAIT) after the circuit station is started.

A circuit station with usage STD can have contact type USERINIT. If it does, your program must send a call request control message to initiate call establishment, as explained in section "Communicating with an STD+ Usage Circuit" on page 110. You receive no indication of whether or not the call is established, so it is not recommended that you specify contact USERINIT with an STD usage circuit. The call request control message is the only control message you can send to an STD usage circuit.

The information used to establish the call for contact type INIT or WAIT comes from the circuit station definition and the X.25 data set, $.SYSX25. The *Design and Installation Guide* explains how to define the required information.

Starting Communication

You can tell that the connection with the other end of the circuit has been made when you receive a message. If you send a message before the connection is made, it may be lost.

You may want to establish some sort of handshaking procedure with the program or device you communicate with. For example, if the circuit is a PVC, you could begin by sending a "hello" message, waiting a few seconds, and checking your message queue to see if you got a response. Continue doing this until you get the response or terminate your program after some number of attempts.

If the circuit is an SVC, you could begin by sending a "hello" message and waiting until your partner responds. The other side would begin by waiting for the message and sending a response when it is received.

You could also wait to send any messages to an SVC circuit station until the call establishment is complete. When the call is connected, bit 15 of the circuit station's Q$STAT field is set to 1.

Here is an example of how to check the circuit station's status; it depends on your program's user station being directly linked to the circuit station.

```
LOCATE   ST,#1                                 FIND YOUR STATION
LOCATE   NA,#1,(Q$DLV,#1),EXIT=NOLINK          FIND ITS LINK
DO       WHILE,((Q$STAT,#1),OFF,15)            IF NOT CONNECTED
  STIMER   500,WAIT                            WAIT 500 MS
ENDDO
           •
           •
```

This method will not work for PVC circuit stations, because bit 15 of Q$STAT indicates that this end of the permanent connection is ready to transmit data, even when the program controlling the other end is not active.

Sending and Receiving Data Messages

Each message that you send or receive begins with an X.25 header, as explained in "Using X.25 Headers" on page 108.

The following example shows how to include the header in messages you send and skip past it in messages you receive.

```
INBUFF   DEFINE  BUFFER,SIZE=256                   INPUT BUFFER
OUTBUFF  DEFINE  BUFFER,SIZE=256                   OUTPUT BUFFER
HELLO    TEXT    'HELLO'                           MESSAGE DATA
*                                                  X.25 HEADER:
         DATA    X'0202'                           TEXT COUNT FIELD
X25HDR   DATA    X'0200'                           HEADER DATA
*
X25LEN   DATA    F'0'                              INPUT HEADER LENGTH
          •
          •
*** SENDING A MESSAGE
         PUT     F,OUTBUFF,X25HDR,OPTION=INITIAL   PUT HEADER
         PUT     F,OUTBUFF,HELLO                   APPEND DATA
         SEND    M,,OUTBUFF                        SEND MESSAGE
          •
          •
*** RECEIVING A MESSAGE
         RECEIVE M,INBUFF                          RECEIVE MESSAGE
         MOVE    X25LEN+1,INBUFF,BYTE              GET HEADER LENGTH
         ADD     INBUFF+B$ADDR,X25LEN,RESULT=#1    SKIP HEADER
         SUB     INBUFF+B$COUNT,X25LEN,RESULT=#2   ADJUST LENGTH
          •
          •
```

Communicating with an STD+ Usage Circuit

You can write one program to communicate with an STD+ usage circuit. The program sends and receives both data and control messages. Link the program and circuit as shown in the section "Communicating with an STD Usage Circuit" on page 108.

You can also write one program to handle the data messages and another to handle the control messages. To do this, set a direct link between the data program and the circuit, a direct link from the control program to the circuit, and an alternate link from the circuit to the control program:

```
> CP LINK data-program circuit BOTH
> CP LINK control-program circuit
> CP LINK circuit control-program ALT
```

You don't need to specify a destination when you send either type of message; they'll be sent to the circuit station. All data messages that come across the circuit will be put on the data-program's message queue as Communications Facility data messages; when you receive a data message, the instruction completes with return code -1. All control packets, including qualified data packets, that come across the circuit will be put on the control-program's message queue as Communications Facility status messages; when you receive a status message, the instruction completes with return code +6.

X.25 control messages are sent to you as status messages whether you use one or two programs to communicate with a circuit. Remember that your program may also receive a status message that is a request to stop, as discussed in "Terminating Your Program" on page 28. The stop request is a 2-byte message that contains either P (stop) or H (halt), followed by a blank.

You must send X.25 control messages as status messages. Use the SEND S instruction if the message is in an EDX text area; use SEND SM if it's in a Communications Facility buffer.

Establishing the Call

If the circuit is a PVC, you don't do anything in your program to establish the call. The connection is always established as long as the circuit station is started. Start communication with a PVC as described in "Starting Communication" on page 109.

If the circuit is a SVC, you have to establish the call by sending and receiving control messages. The procedure varies depending on the type of contact defined for the circuit station.

When the contact type is WAIT, the circuit waits for a call from another DTE. When an incoming call packet comes across the circuit, the X.25 IOCP sends your program an incoming call control message. You can either:

- Send a call accept control message to accept the call. You can then begin data transfer.

- Send a clear request control message to reject the call. You will receive a clear confirmation control message. You should then wait for another incoming call control message.

When the contact type is INIT, the X.25 IOCP sends a call request packet when the circuit station is started. The address it calls is the one defined for the circuit station's call ID in data set $.SYSX25.

One of the following will occur:

- The other side accepts the call. You will receive a call connected control message. You can then begin data transfer.

- The other side rejects the call. You will receive a clear indication control message. The IOCP then either stops the circuit station if it is in STOP mode, or tries again to establish the call if it is in RETRY mode. When you receive a clear indication, you should either terminate the program or wait again for the call to be established, depending on the circuit station's mode. If you don't know its mode, you can locate the circuit station and check bit 9 of field Q$DVD; 0=STOP mode, 1=RETRY mode.

- There is no entry for the circuit station's call ID in $.SYSX25, the entry is invalid, or $.SYSX25 is not accessible. You will receive an error indication control message. You should terminate your program. Then stop the circuit station, correct $.SYSX25, and start the circuit station and your program(s).

When the contact type is USERINIT, you must send a call request control message to initiate the call. One of the following will occur:

- The other side accepts the call. You will receive a call connected control message. You can then begin data transfer.

- The call request control message is invalid or the other side rejects the call. You will receive a clear indication control message with information that indicates why the call was rejected. If the call request was invalid, you'll need to correct your program. If the other side rejected the call, you can either either send another call request control message or terminate the program.

- If you used an abbreviated address in your call request control message, there is no entry for the circuit station's call ID in $.SYSX25, the entry is invalid, or $.SYSX25 is not accessible. You will receive an error indication control message. Proceed as described for contact type INIT.

Once the call is established, you can send and receive data messages with an STD+ circuit just like you do with an STD circuit, as described in "Sending and Receiving Data Messages" on page 109.

Using Network Facilities

You can use network facilities that are subscribed to with the X.25 network carrier during call establishment. You can define certain X.25 network facilities in contact type INIT or USERINIT circuit stations, using $.CONFIG or the CP F FAC command. These facilities include fast select, reverse charge, closed user group, bilateral closed user group, and RPOA. Facilities defined in the station are always included in the call request packets sent for the circuit.

You can also include X.25 and non-X.25 facilities in the call request control message you send to contact type USERINIT circuit stations. The format of the facility field in the call request control message is identical with the format of the facility field in call request packets described in CCITT Recommendation X.25. Each facility consists of a code and a parameter. The facility code specifies which facility is being used, and the facility parameter specifies the value of the facility. Figure 27 lists some examples of X.25 network facility codes and parameters.

Code (Hex)	Keyword	Values	Sample (Hex)	Description
01				This code includes 3 facilities:
	REV	Bit 7 = 1	0101	Reverse the charges
	FS	Bits 0,1 = 10	0180	Fast select facility—call request and clear request or call accept can carry user data for delivery before call is established.
	FSR	Bits 0,1 = 11	01C0	Fast select, restriction on response—like fast select, but call must be cleared, may not be accepted.

Figure 27 (Part 1 of 2). X.25 Network Facilities

Code (Hex)	Keyword	Values	Sample (Hex)	Description
02	...	Coded bps	0288	Class throughput negotiation— not looked at by X.25 IOCP, but will be passed to network.
03	CUG	Number	0302	Closed user group number— to use this, you must have subscribed to at least one closed user group in your network.
41	BCUG	Number	410365	Bilateral closed user group— A special kind of user group.
42	...	Coded sizes	4208	Packet size negotiation— not looked at by the X.25 IOCP but will be passed on to network.
43	...	Sizes	4304	Window size negotiation—not looked at by the X.25 IOCP but will be passed on to network.
44	RPOA	4 BCD digits	440122	RPOA transit network data network ID code.

Figure 27 (Part 2 of 2). X.25 Network Facilities

You can include non-X.25 facilities in your call request control message by separating the X.25 facilities from the non-X.25 facilities with a 2-byte facility marker. The facility code in the marker must be 0. Note that X.25 facilities must always precede the non-X.25 facilities.

You can include any X.25 facilities in your call request—not just the ones you can define in circuit station definitions. If you include facilities in your call request control message, and the same facilities are defined in the circuit station definition, the values in the call request control message are used instead of the values defined in the station. If the station has facilities that are not included in the call request control message, they are added to the call request packet before it is sent. The length of the facility field—including all facility codes and parameters—cannot exceed 63 bytes.

Sending and Receiving Control Messages

When you send a control message, the X.25 IOCP builds a corresponding X.25 control packet and sends it across the circuit. When an X.25 control packet comes across the circuit, the IOCP builds a corresponding control message and sends it to your program (that is, to the circuit station's direct or alternate link). The control messages closely resemble their corresponding X.25 control packets, but they are not identical.

The rest of this section describes the control messages you can send or receive. You can use them with either PVCs or SVCs, unless noted otherwise. The format of the control messages is given in section "X.25 Control Messages" on page 116.

Sending Control Messages

The control messages that you can send are:

Call Accept
Accepts a call on an SVC whose contact type is WAIT. You send it as a response to an incoming call control message. The message may optionally contain the called and calling X.25 network addresses; you can obtain these from the incoming call control message. It may contain negotiable network facilities—facilities specified by the caller that the accepter must acknowledge or that the accepter can override. If the fast select facility is used during call establishment, the message may also contain a protocol identifier and user data.

Call Request
Initiates a call on an SVC whose contact type is USERINIT. The message contains the X.25 network address to be called, or an abbreviated address with the call ID of the address to be called. It may optionally contain the network facilities to be used for the call, a protocol identifier, and user data.

Clear Request
Rejects a call on an SVC whose contact type is WAIT. You send it as a response to an incoming call control message. If the fast select facility is used during call establishment, the message may also contain user data.

You can also send a clear request control message to clear an established call on an SVC with any contact type. Once the call has been cleared, a new call can be established.

After you send a clear request control message for either reason, you'll receive a clear confirmation control message, if the other side confirms the clear within the time prescribed by CCITT Recommendation X.25. If the time limit is exceeded, the X.25 I/O control program closes the X.25 circuit and you'll receive a clear indication control message.

Interrupt
Sends 1 byte of user data to the other side without flow control. After you send an interrupt control message, you'll receive an interrupt confirmation control message. You can't send another interrupt until you receive an interrupt confirmation; you can send data messages.

Passthrough
Sends user-defined control information as qualified data packets (data packets with the Q-bit on). You can use passthrough messages to send any sort of data that you want your partner to receive as a control message. Their length is restricted only by the buffer size of the line station with which the circuit station is associated.

Reset Request
Reinitializes the circuit. Packets in transit may be discarded. When you send a reset request control message, you'll receive a reset confirmation control message if the other side confirms the reset within the time prescribed by CCITT Recommendation X.25. If the time limit is exceeded, the X.25 I/O control program closes the X.25 circuit and you'll receive a reset (PVC) or clear (SVC) indication control message.

Receiving Control Messages

The control messages that you can receive are:

Call Connected
Indicates that the other DTE has accepted a call on an SVC you (contact USERINIT) or the X.25 I/O control program (contact INIT) initiated. The message may optionally contain the called and calling X.25 network addresses or negotiable network facilities. If the fast select facility is used during call establishment, the message may also contain a protocol identifier and user data.

Clear Confirmation
Confirms that the SVC has been cleared as a result of a clear request control message you sent. The message contains no data.

Clear Indication
Indicates that the other DTE has rejected a call on an SVC you (contact USERINIT) or the X.25 IOCP (contact INIT) initiated. The message contains codes that indicate why the call was rejected. If the fast select facility is used during call establishment, the message may also contain user data.

You also receive a clear indication control message when the SVC has been cleared as a result of a clear request by the other side, or as a result of an error detected by the X.25 network or $XHCS. The message contains codes that indicate why the circuit was cleared.

Error Indication
Indicates that the X.25 I/O control program rejected a control message you sent because the message is invalid, there is something wrong with the circuit station, or there is some problem with data set $.SYSX25. You also receive an error indication control message when the IOCP cannot initiate a call (contact INIT) because of a problem with the circuit station or $.SYSX25. The message contains codes that identify the error.

Incoming Call
Indicates that a call has been initiated by the other side on an SVC whose contact type is WAIT. The message contains the called X.25 network address. It may optionally contain the calling X.25 network address, the network facilities to be used for the call, a protocol identifier, and user data.

You must send a call accept control message if you accept the call or a clear request control message if you reject the call.

Interrupt
Contains 1 byte of user data sent by the other side without flow control.

Interrupt Confirmation
Confirms that the other side received an interrupt packet as the result of an interrupt control message you sent. The message contains no data.

Passthrough
Contains user-defined control information sent by the other side in a data packet with the Q-bit set.

Reset Confirmation
Confirms that the circuit has been reset as a result of a reset request control message you sent. The message contains no data.

Reset Indication
Indicates that the circuit has been reset as a result of a reset request by the other side, or an error detected by the X.25 network or $XHCS. The message contains codes that indicate why the circuit was reset.

X.25 Control Messages

This section gives the format of the X.25 control messages you can send or receive when you use an STD+ usage circuit. Each message begins with an X.25 header, as explained in "Using X.25 Headers" on page 108. With the present version of X.25 support, the message type code is in bits 2-7 of the second byte of the header; bits 0-1 (the format identifier) are 00.

In the following formats, the numbers shown above specific fields represent bytes. Certain fields have no numbers above them; this means that the field length is variable.

Call Accept

The call accept control message accepts a call on an SVC whose contact type is WAIT. You send it as a response to an incoming call control message.

1	2	3	4			
hl	*mc*	*al*	*called*	*calling*	*fl*	*facilities*

hl
: is the length, in bytes, of the X.25 header, For the present version of X.25 support, this field is X‘02’.

mc
: is the format identifier and the message type code, X‘0F’.

al
: is the number of BCD digits in the next two fields. The first 4 bits are the number of BCD digits in the calling address (0-15). The last 4 bits are the number of BCD digits in the called address (0-15). The field is optional if all following fields are omitted.

called
: is the called X.25 network address, specified as one to 15 BCD digits (one binary digit per half-byte). If the address is an odd number of digits, the first digit of the calling address is in the same byte as the last digit of the called address. The field is optional.

calling
: is the calling X.25 network address, specified as one to 15 BCD digits (one binary digit per half-byte). The field is optional.

fl
: is a 1-byte field that contains the length, in bytes, of the next field. The field is optional if the next field is omitted.

facilities
: is the negotiable network facilities to be used for the call. See “Using Network Facilities” on page 112 for the format of this field. The field is optional.

Example

```
X'020F0812345678'
```

where 12345678 is the 8-digit called address.

Call Accept with Fast Select Facility

The call accept control message accepts a call on an SVC whose contact type is WAIT. You send it as a response to an incoming call control message that requested the fast select facility.

1	2	3	4					
hl	*mc*	*al*	*called*	*calling*	*fl*	*facilities*	*protid*	*user-data*

hl
is the length, in bytes, of the X.25 header. For the present version of X.25 support, this field is X'02'.

mc
is the format identifier and the message type code, X'0F'.

al
is the number of BCD digits in the next two fields. The first 4 bits are the number of BCD digits in the calling address (0-15). The last 4 bits are the number of BCD digits in the called address (0-15). The field is required.

called
is the called X.25 network address, specified as one to 15 BCD digits (one binary digit per half-byte). If the address is an odd number of digits, the first digit of the calling address is in the same byte as the last digit of the called address. The field is optional.

calling
is the calling X.25 network address, specified as one to 15 BCD digits (one binary digit per half-byte). The field is optional.

fl
is a 1-byte field that contains the length, in bytes, of the next field. The field is required; specify 0 if the next field is omitted.

facilities
is the negotiable network facilities to be used for the call. See "Using Network Facilities" on page 112 for the format of this field. The field is optional.

protid
is the protocol identifier, a 4-byte user-defined value.

The first 2 bits are significant to public data networks. Depending on their value, the protocol identifier and the user data field will be used in accordance with the specifications of the following:

00 - Recommendation X.29
01 - Network Administrations
10 - International User Bodies
11 - No constraints

A protocol identifier whose first 2 bits are other than 11 may cause a protocol to be implemented within public data networks.

The field is optional if the following field is omitted.

user-data
is user data such as a password or function selection information. The data can be up to 124 bytes long.

Example

```
X'020F081234567800C0001234',C'INVENTORY'
```

where 12345678 is the 8-digit called address; there are no negotiated facilities; C0001234 is the protocol identifier; and INVENTORY is the user data.

Call Connected

The call connected control message indicates that a call on an SVC has been accepted. You receive it as a response to a call initiated by you (contact USERINIT) or by the X.25 IOCP (contact INIT).

1	2	3	4			
hl	*mc*	*al*	*called*	*calling*	*fl*	*facilities*

hl
: is the length, in bytes, of the X.25 header. For the present version of X.25 support, this field is X'02'.

mc
: is the format identifier and the message type code, X'0F'.

al
: is the number of BCD digits in the next two fields. The first 4 bits are the number of BCD digits in the calling address (0-15). The last 4 bits are the number of BCD digits in the called address (0-15). The field is required; specify 0 if the next two fields are omitted.

called
: is the called X.25 network address, specified as one to 15 BCD digits (one binary digit per half-byte). If the address is an odd number of digits, the first digit of the calling address is in the same byte as the last digit of the called address. The field is optional.

calling
: is the X.25 network address that initiated the call, specified as one to 15 BCD digits (one binary digit per half-byte). The field is optional.

fl
: is a 1-byte field that contains the length, in bytes, of the next field. The field is required; specify 0 if the next fields is omitted.

facilities
: is the negotiable network facilities to be used for the call. See "Using Network Facilities" on page 112 for the format of this field. The field is optional.

Example

```
X'020F081234567800'
```

where 12345678 is the 8-digit called address and there are no negotiated facilities.

Call Connected with Fast Select facility

The call connected control message indicates that a call on an SVC has been accepted. You receive it as a response to a call initiated by you (contact USERINIT) or by the X.25 IOCP (contact INIT), which requested the fast select facility.

1	2	3	4					
hl	*mc*	*al*	*called*	*calling*	*fl*	*facilities*	*protid*	*user-data*

hl
: is the length, in bytes, of the X.25 header. For the present version of X.25 support, this field is X'02'.

mc
: is the format identifier and the message type code, X'0F'.

al
: is the number of BCD digits in the next two fields. The first 4 bits are the number of BCD digits in the calling address (0-15). The last 4 bits are the number of BCD digits in the called address (0-15). The field is required.

called
: is the called X.25 network address, specified as one to 15 BCD digits (one binary digit per half-byte). If the address is an odd number of digits, the first digit of the calling address is in the same byte as the last digit of the called address. The field is optional.

calling
: is the X.25 network address that initiated the call, specified as one to 15 BCD digits (one binary digit per half-byte). The field is optional.

fl
: is a 1-byte field that contains the length, in bytes, of the next field. The field is required; specify 0 if the next field is omitted.

facilities
: is the negotiable network facilities to be used for the call. See "Using Network Facilities" on page 112 for the format of this field. The field is optional.

protid
: is the protocol identifier, a 4-byte user-defined value.

The first 2 bits are significant to public data networks. Depending on their value, the protocol identifier and the user data field will be used in accordance with the specifications of the following:

00 - Recommendation X.29
01 - Network Administrations
10 - International User Bodies
11 - No constraints

A protocol identifier whose first 2 bits are other than 11 may cause a protocol to be implemented within public data networks.

The field is optional if the following field is omitted.

user-data
is user data such as a password or function selection information. The data can be up to 124 bytes long.

Example

```
X'020F081234567800C0001234',C'INVENTORY'
```

where 12345678 is the 8-digit called address; there are no negotiated facilities; C0001234 is the protocol identifier; and INVENTORY is the user data.

Call Request

The call request control message initiates a call on an SVC whose contact type is USERINIT.

1	2	3	4				
hl	*mc*	*al*	*called*	*fl*	*facilities*	*protid*	*user-data*

hl
: is the length, in bytes, of the X.25 header. For the present version of X.25 support, this field is X'02'.

mc
: is the format identifier and the message type code, X'0B'.

al
: is the number of digits in the address to be called, including the period if an abbreviated address is specified. The field is required.

called
: is the X.25 network address to be called, specified in one of two ways:

- an actual X.25 network address—up to 15 EBCDIC digits. The X.25 IOCP converts the address to BCD digits.
- an abbreviated network address—a period followed by a 2-digit EBCDIC call ID. The IOCP looks up the call ID in data set $.SYSX25 and initiates the call to the associated X.25 network address.

fl
: is a 1-byte field that contains the length, in bytes, of the next field. This field is required; specify 0 if the next field is omitted.

facilities
: is the negotiable network facilities to be used for the call. See "Using Network Facilities" on page 112 for the format of this field. The field is optional.

protid
: is the protocol identifier, a 4-byte user-defined value that can be used to decide whether or not a call from a particular X.25 network address will be accepted.

The first 2 bits are significant to public data networks. Depending on their value, the protocol identifier and the user data field will be used in accordance with the specifications of the following:

00 - Recommendation X.29
01 - Network Administrations
10 - International User Bodies
11 - No constraints

A protocol identifier whose first 2 bits are other than 11 may cause a protocol to be implemented within public data networks.

The field is optional if the following field is omitted.

user-data
is user data such as a password or function selection information. The data can be up to 12 bytes, or up to 124 bytes if the fast select facility is being used. The field is optional.

Example

```
X'020B06F1F1F2F2F3F305018141 5566C0001234',C'PASSWORDCSAD'
```

where 112233 is the 6-digit called address in EBCDIC; in the 5-byte facility field, 0181 is the fast select and reverse charge facilities, and 415566 is the bilateral closed user group facility with a group number of 5566; C0001234 is the protocol identifier; and PASSWORDCSAD is the user data field.

Note that there is no calling address field in a call request control message. If you want a calling address in the call request, use $.CONFIG or the CP F ADDR command to define the address in the circuit's controlling line station. If the line station contains a calling address, the X.25 I/O control program includes it in all call requests it sends out on the line's circuits.

Clear Confirmation

The clear confirmation control message confirms that the SVC has been cleared as a result of a clear request control message you sent.

1	2
hl	*mc*

hl

is the length, in bytes, of the X.25 header. For the present version of X.25 support, this field is X'02'.

mc

is the format identifier and the message type code, X'17'.

Clear Indication

The clear indication control message indicates that the SVC has been cleared as a result of a clear request by the other side, an error detected by the X.25 network or $XHCS, or because a CP P command issued for the circuit station.

1	2	3	4
hl	*mc*	*ca*	*di*

hl

is the length, in bytes, of the X.25 header. For the present version of X.25 support, this field is X'02'.

mc

is the format identifier and the message type code, X'13'.

ca

is the cause code. It contains one of the following values:

X'00' - DTE originated
X'01' - Number busy
X'03' - Invalid facility request
X'05' - Network congestion
X'09' - Out of order
X'0B' - Access barred
X'0D' - Not obtainable
X'11' - Remote procedure error
X'13' - Local procedure error
X'15' - RPOA out of order
X'19' - Reverse charging not subscribed
X'21' - Incompatible destination
X'29' - Fast select acceptance not subscribed
X'F0' - Error on line
X'FF' - Circuit station was stopped

di

is the diagnostic code. See Figure 28 on page 152 for a list of diagnostic codes and their meanings.

Example

```
X'02130300'
```

where the cause code, 03, means you requested an invalid facility in the call request packet sent on your circuit, and 00 is the diagnostic code that indicates there is no additional information.

Clear Indication with Fast Select Facility

The clear indication control message is used in conjunction with the fast select facility to indicate that the SVC has been cleared as a result of a clear request by the other side, an error detected by the X.25 network or $XHCS, or because of a CP P command issued for the circuit station.

1	2	3	4	5					
hl	*mc*	*ca*	*di*	*al*	*called*	*calling*	*fl*	*facilities*	*user-data*

hl

is the length, in bytes, of the X.25 header. For the present version of X.25 support, this field is X'02'.

mc

is the format identifier and the message type code, X'13'.

ca

is the cause code. It contains one of the following values:

X'00' - DTE originated
X'01' - Number busy
X'03' - Invalid facility request
X'05' - Network congestion
X'09' - Out of order
X'0B' - Access barred
X'0D' - Not obtainable
X'11' - Remote procedure error
X'13' - Local procedure error
X'15' - RPOA out of order
X'19' - Reverse charging not subscribed
X'21' - Incompatible destination
X'29' - Fast select acceptance not subscribed
X'F0' - Error on line
X'FF' - Circuit station was stopped

di

is the diagnostic code. See Figure 28 on page 152 for a list of diagnostic codes and their meanings.

al

is the number of BCD digits in the next two fields. The first 4 bits are the number of BCD digits in the calling address (0-15). The last 4 bits are the number of BCD digits in the called address (0-15). The field is optional if all of the following fields are omitted. At present, the field is 0 if any of the following fields are specified.

called

is the called X.25 network address, specified as one to 15 BCD digits (one binary digit per half-byte). At present, omit this field.

calling

is the X.25 network address that initiated the call, specified as one to 15 BCD digits (one binary digit per half-byte). At present, omit this field.

fl
: is a 1-byte field that contains the length, in bytes, of the facilities field. The field is optional if the user-data field is omitted. At present, the field is 0 if the user-data field is specified.

facilities
: is the facilities field. At present, this field is omitted.

user-data
: is up to 128 bytes of user data. The field is optional.

Example

```
X'021300000000',C'CUSTOMER SMITH OK FOR PURCHASE'
```

where:

00 (DTE originated) is the cause code;
00 (no further information) is the diagnostic code;
00 is the address lengths;
00 is the facilities length; and
CUSTOMER SMITH OK FOR PURCHASE is the clear user data.

Clear Request

The clear request control message clears an SVC. You send it either in response to an incoming call control message or to clear an established call.

1	2	3	4
hl	*mc*	*ca*	*di*

hl

is the length, in bytes, of the X.25 header. For the present version of X.25 support, this field is X‘02’.

mc

is the format identifier and the message type code, X‘13’.

ca

is the cause code. Specify X‘00’ (DTE originated) or omit the field.

di

is the diagnostic code. See Figure 28 on page 152 for a list of diagnostic codes and their meanings. The field is optional.

Example

```
X'021300'
```

where 00 is the cause code for DTE originated.

Clear Request with Fast Select Facility

The clear request control message clears an SVC. You send it in response to an incoming call control message which specified the fast select facility if you wish to clear the the call. You must send it in response to an incoming call which specified the fast select facility with restriction on response.

1	2	3	4	5					
hl	*mc*	*ca*	*di*	*al*	*called*	*calling*	*fl*	*facilities*	*user-data*

hl

is the length, in bytes, of the X.25 header. For the present version of X.25 support, this field is X'02'.

mc

is the format identifier and the message type code, X'13'.

ca

is the cause code. Specify X'00' (DTE originated).

di

is the diagnostic code. See Figure 28 on page 152 for a list of diagnostic codes and their meanings. The field is optional if all of the following fields are omitted.

al

is the number of BCD digits in the next two fields. The field is optional if all of the following fields are omitted. Specify 0 if any of the following fields are present.

called

is the called X.25 network address, specified as one to 15 BCD digits (one binary digit per half-byte). At present, omit this field.

calling

is the X.25 network address that initiated the call, specified as one to 15 BCD digits (one binary digit per half-byte). At present, omit this field.

fl

is a 1-byte field that contains the length, in bytes, of the facility field. The field is optional if the user-data field is omitted. Specify 0 if user-data is present.

facilities

is the facilities field. At present, this field is omitted.

user-data

is up to 128 bytes of user data. The field is optional.

Example

```
X'021300000000',C'CUSTOMER SMITH OK FOR PURCHASE'
```

where:

00 (DTE originated) is the cause code;
00 (no further information) is the diagnostic code;
00 is the address lengths;
00 is the facilities length;
and CUSTOMER SMITH OK FOR PURCHASE is the clear user data.

Error Indication

The error indication control message indicates that the X.25 IOCP rejected a control message you sent or could not initiate a call on an SVC (contact INIT).

1	2	3	4
hl	*mc*	*al*	*cd*

hl

is the length, in bytes, of the X.25 header. For the present version of X.25 support, this field is X'02'.

mc

is the format identifier and the message type code, X'35'.

ca

is the cause code. It contains one of the following values:

X'32' - Interrupt data is wrong length
X'33' - Received unknown control message
X'35' - Clear is not allowed on PVC
X'39' - Reset data is too long
X'44' - Interrupt is already pending
X'64' - Error in $.SYSX25
X'90' - Circuit is in invalid state
X'91' - SVC has invalid contact type

cd

is additional information for the following cause codes (it is omitted for the other cause codes):

33: The unknown message code

64:

X'02'—call ID not found in $.SYSX25 or entry for call ID is invalid

X'03'—Disk I/O error during access to $.SYSX25

90: The invalid circuit state

91: The invalid contact type

Example

```
X'02356403'
```

where 64 is the error cause code for a $.SYSX25 data set error, and 03 means it was a hardware error.

Incoming Call

The incoming call control message indicates that a call has been initiated by the other DTE on an SVC whose contact type is WAIT.

1	2	3	4					
hl	*mc*	*al*	*called*	*calling*	*fl*	*facilities*	*protid*	*user-data*

hl
: is the length, in bytes, of the X.25 header. For the present version of X.25 support, this field is X'02'.

mc
: is the format identifier and the message type code, X'0B'.

al
: is the number of BCD digits in the next two fields. The first 4 bits are the number of BCD digits in the calling address (0-15). The last 4 bits are the number of BCD digits in the called address (0-15).

called
: is the called X.25 network address, specified as one to 15 BCD digits (one binary digit per half-byte). If the address is an odd number of digits, the first digit of the calling address is in the same byte as the last digit of the called address.

calling
: is the X.25 network address that initiated the call, specified as one to 15 BCD digits (one binary digit per half-byte).

fl
: is a 1-byte field that contains the length, in bytes, of the next field. It contains 0 if the next field is omitted.

facilities
: is the negotiable network facilities to be used for the call. See "Using Network Facilities" on page 112 for the format of this field. The field is optional.

protid
: is the protocol identifier, a 4-byte user-defined value that can be used to decide whether or not a call from a particular X.25 network address will be accepted.

The first 2 bits are significant to public data networks. Depending on their value, the protocol identifier and the user data field will be used in accordance with the specifications of the following:

00 - Recommendation X.29
01 - Network Administrations
10 - International User Bodies
11 - No constraints

A protocol identifier whose first 2 bits are other than 11 may cause a protocol to be implemented within public data networks.

The field is optional if the following field is omitted.

user-data
: is user data such as a password or function selection information. The data can be up to 12 bytes, or up to 124 bytes if the fast select facility is being used. The field is optional.

Example

```
X'020F65112231234560Q201C1'
```

where 11223 is the 5-digit called address; 123456 is the 6-digit calling address; and in the 2-byte facility field, 01C1 is the fast select restriction on response and reverse charge facilities.

Interrupt

The interrupt control message contains 1 byte of user-defined control information. You can send or receive an interrupt control message.

1	2	3
hl	*mc*	*da*

hl
: is the length, in bytes, of the X.25 header. For the present version of X.25 support, this field is X'02'.

mc
: is the format identifier and the message type code, X'23'.

da
: is the byte of interrupt data.

Example

```
X'0223D2'
```

where D2 is the byte of interrupt data.

Interrupt Confirmation

The interrupt confirmation control message confirms that the other side received the interrupt control message you sent.

1	2
hl	*mc*

hl
: is the length, in bytes, of the X.25 header. For the present version of X.25 support, this field is X‘02’.

mc
: is the format identifier and the message type code, X‘27’.

Passthrough

The passthrough control message contains user-defined control information. The message flows on the X.25 circuit in qualified data packets. You can send or receive a passthrough control message.

1	2	3
hl	*mc*	*data*

hl
: is the length, in bytes, of the X.25 header. For the present version of X.25 support, this field is X‘02’.

mc
: is the format identifier and the message type code, X‘00’.

data
: is the user-defined data.

Example

```
X'0200',C'CLASSIFIED: SMITH'S CREDIT RATING IS VERY GOOD'
```

where CLASSIFIED: SMITH’S CREDIT RATING IS VERY GOOD is the data to be sent as qualified data.

Reset Confirmation

The reset confirmation control message confirms that the circuit has been reset as a result of a reset request control message you sent.

1	2
hl	*mc*

hl

is the length, in bytes, of the X.25 header. For the present version of X.25 support, this field is X'02'.

mc

is the format identifier and the message type code, X'1F'.

Reset Indication

The reset indication control message indicates that the circuit has been reset as a result of a reset request by the other side, an error detected by the X.25 network or $XHCS, or because of a CP P command issued for a PVC circuit station.

1	2	3	4
hl	*mc*	*ca*	*di*

hl

is the length, in bytes, of the X.25 header. For the present version of X.25 support, this field is X'02'.

mc

is the format identifier and the message type code, X'1B'.

ca

is the cause code. It contains one of the following values:

X'00' - DTE originated
X'01' - Out of order (PVCs only)
X'03' - Remote procedure error
X'05' - Local procedure error
X'09' - Remote DTE operational (PVCs only)
X'0F' - Network operational (PVCs only)
X'11' - Incompatible destination
X'F0' - Error on line (PVCs only)
X'FF' - Circuit station was stopped (PVCs only)

di

is the diagnostic code. See Figure 28 on page 152 for a list of diagnostic codes and their meanings. The field is optional.

Example

```
X'021B0900'
```

where the cause code, 09, means the other end of the PVC is active and ready to receive messages, and the diagnostic code, 00, means there is no additional information.

Reset Request

The reset request control message reinitializes the circuit by resetting the windows on each side to zero.

1	2	3	4
hl	*mc*	*ca*	*di*

hl

is the length, in bytes, of the X.25 header. For the present version of X.25 support, this field is X'02'.

mc

is the format identifier and the message type code, X'1B'.

ca

is the cause code. Specify X'00' (DTE originated) or omit the field.

di

is the diagnostic code. See Figure 28 on page 152 for a list of diagnostic codes and their meanings. The field is optional.

Example

```
X'021B00'
```

where 00 is the cause code for DTE originated.

Diagnostic Code **Meaning**

Decimal	Hex	
0	00	No diagnostic information
1	01	Invalid P(S)
2	02	Invalid P(R)
16	10	Packet type invalid
17	11	Packet type invalid for state R1
18	12	Packet type invalid for state R2
19	13	Packet type invalid for state R3
20	14	Packet type invalid for state P1
21	15	Packet type invalid for state P2
22	16	Packet type invalid for state P3
23	17	Packet type invalid for state P4
24	18	Packet type invalid for state P5
25	19	Packet type invalid for state P6
26	1A	Packet type invalid for state P7
27	1B	Packet type invalid for state D1
28	1C	Packet type invalid for state D2
29	1D	Packet type invalid for state D3
32	20	Packet not allowed
33	21	Unidentifiable packet type
34	22	Incoming call on one way logical channel
35	23	Invalid packet type on a permanent virtual circuit
36	24	Packet received on an unassigned logical channel
37	25	Reject packet not supported
38	26	Packet too short
39	27	Packet too long
40	28	Invalid general format identifier
41	29	Restart packet received with nonzero in bits 1-4, 9-16
42	2A	Packet type not compatible with facility
43	2B	Unauthorized interrupt confirmation
44	2C	Unauthorized interrupt
48	30	Timer expired
49	31	Timer expired for incoming call
50	32	Timer expired for clear indication
51	33	Timer expired for reset indication
52	34	Timer expired for restart indication

Figure 28 (Part 1 of 2). Diagnostic Codes

64	40	Call set-up problem
65	41	Facility code not allowed
66	42	Facility parameter not allowed
67	43	Invalid called address
68	44	Invalid calling address
80	50	Not assigned
161	A1	Invalid M-bit on non-full Data packet
174	AE	Invalid Q-bit on packet

Figure 28 (Part 2 of 2). Diagnostic Codes

Creating a Command

The Communications Facility command processor consists of an initial task (task CP in the control program, S$CF) plus a set of independent command-processing programs (one per command). To add a new command to the Communications Facility, you must:

- Decide what functions the command is to provide.
- Define the command's syntax.
- Name your command-processing program so the Communications Facility will recognize it as a program that processes a command.
- Code your command-processing program.
- Add a description of the command to the module S$CPHELP, which processes the CP HELP command. (This step is not strictly required, but is recommended.)

The rest of this chapter discusses the following topics, which you need to consider as you code your command-processing program:

- Naming your program.
- Retrieving the command parameters the user entered.
- Getting information from, and writing information to, the network configuration data set ($.SYSNET).
- Getting information from, and writing information to, other Communications Facility data sets.
- Sending messages to indicate successful completion or error conditions.
- Sending a completion code upon termination.
- Avoiding deadlocks.
- Ensuring that your program will load.

Example Command-Processing Program

Example command-processing program S$CPBIT, shown in Figure 29 on page 159, illustrates the steps described in the rest of this chapter. We'll be referring to this example throughout.

S$CPBIT processes the BIT command (which is *not* part of the Communications Facility). The BIT command modifies a station by setting a bit in the status word (Q$STAT) on or off. The status word is modified in the system configuration data set, $.SYSNET. If the station is started, BIT also modifies the status word in the station block.

The syntax of the BIT command is:

CP BIT *station-name bit-number* [*bit-value*]

where:

station-name
is the name of the station.

bit-number
indicates which bit (0-15) is to be modified.

bit-value
is the value to which the bit is to be set (0 or 1); the default is 1.

Naming Your Program

When task CP receives a command message, it moves the message to a Communications Facility buffer and gets the first field, the command name. It then issues an instruction to load program $.CP*xxxx*, where *xxxx* is the 1- to 4-character command name. Therefore, the load module name of your command-processing program must be $.CP*xxxx*. To conform to Communications Facility naming conventions, the source module name should be S$CP*xxxx*.

Retrieving the Command's Parameters

The address of the buffer that contains the command is in field Q$BADDR of the message dispatcher's station block. Retrieve the address as shown in statements 150-160 of the example program. The buffer header fields point to the character following the blank that delimits the command name. Use the GET F instruction to retrieve each parameter, specifying a blank delimiter. Subroutine GETPARM in the example program (statements 1440-1690) retrieves parameters, allowing for multiple blanks between parameters.

Gaining Access to $.SYSNET

When the Communications Facility is started, it opens $.SYSNET using a data set control block in the common area. Specify DS=$$ in your PROGRAM statement, and modify your data set control block as shown in statements 170-240 of the example program.

$.SYSNET is a partitioned data set, with a 1-record member for each defined station. The format of a member is the same as the format of a station block. Use a subroutine like FINDMEM in the example program (statements 2010-2400) to determine whether or not a station is defined and to obtain its record number.

Updating $.SYSNET

While a station that has a disk queue is active, disk queuing control information is maintained in its $.SYSNET member. If you modify that member while the station is active, you must enqueue on the member to prevent concurrent updating by your program and the disk queuing programs. Do the enqueuing as shown in statements 900-990 of the example program. The QCB used by the disk queuing programs is in a file control block whose address is in the station block.

You must dequeue before your program terminates. Subroutine DEQNET (statements 1870-1990) of the example program does the dequeuing. Note that, before issuing the DEQ, it ensures that this task is the current owner of the resource. EDX allows any task to release a shared resource.

Gaining Access to Other Data Sets

All Communications Facility data sets reside on one volume, the one from which the control program is loaded. If your program requires access to other data sets (the example program doesn't), you can address the Communications Facility volume by specifying DS=(*dsname*,##) in your PROGRAM statement.

Logging Errors and Successful Completion

You must send a message to the Communications Facility log before your program terminates. The message is either an informational message that indicates successful completion or an error message that describes the error. Error messages must be CP messages, because the completion code reflects the CP message number, as described under "Sending a Completion Code." Use only existing CP error messages. There is no assurance that currently unused CP messages will not be used in the future. If you need a new information message, either specify its full text in the SEND L instruction or use your own member of $.SYSMSG.

The example program uses these existing error messages:

CP03 *station-name* $.SYSNET I/O ERROR
CP09 *parameter* IS INVALID
CP12 *station-name* NOT DEFINED IN $.SYSNET
CP44 *parameter* IS OMITTED

and this information message:

UM01 *station-name* STATUS MODIFIED

Sending a Completion Code

The PROGSTOP instruction that terminates your program must specify a negative completion code, either -1 for successful completion or the negative of the CP error message number.

After task CP loads your program, it waits on an event which is posted either if the load fails or when your program issues a PROGSTOP. If the completion code is positive, task CP assumes a load error and issues this message:

CP59 *command* ERROR ON LOAD

Subroutine ERROR (statements 1710-1850) of the example program issues an error message and sets the corresponding completion code.

Avoiding Deadlocks

The command processor is a single-thread operation. Task CP loads a command-processing program and waits for it to complete before receiving the next command message. Therefore, your program must not wait on an event that may not complete. In particular, you must not use the instruction SEND CP with ACK=YES. Your program should include a task error-exit routine, as shown in statements 1380-1420 of the example program.

Ensuring Your Program Will Load

Command-processing programs are loaded into the same partition as the Communications Facility control program. When the Communications Facility is started, task CP loads the largest command-processing program ($.CPS) to ensure that there is enough space for it. Your program must be no larger than $.CPS, which is approximately 6K; its exact size is in the *Design and Installation Guide.*

```
************************************************************************
* S$CPBIT: COMMUNICATIONS FACILITY EXAMPLE COMMAND PROCESSOR.          *
************************************************************************
*
S$CPBIT  PROGRAM  START,500,DS=$$,ERRXIT=TEECB
         COPY     S$CFEQU
         COPY     DSCBEQU
*
***********************************************************************
* INITIALIZATION.                                                     *
***********************************************************************
*
START    EQU   *
         MOVE  ENDCODE,-1            INITIALIZE COMPLETION CODE
         LOCATE ST,#1,'$.DISP'       GET ADDR OF $.DISP STATION BLOCK
         MOVE  CPBFAD,(Q$BADDR,#1)   GET ADDR OF CP BUFFER
*                                    SET UP $.SYSNET DSCB
         GET   A,#1,CSXTABLE
         MOVE  DS1+$DSCBORN,(-54,#1),2  ORIGIN
         AND   DS1+$DSCBORN,X'2000',RESULT=DS1+$DSCBFLG  FH INDICATOR
         AND   DS1+$DSCBORN,X'DFFF'     TURN OFF FH INDICATOR, IF ON
         MOVE  DS1+$DSCBLNG,(-50,#1),2  LENGTH
         MOVE  DS1+$DSCBVDE,(-46,#1)    VDE ADDRESS
         MOVE  DS1+$DSCBNEX,0,2         NEXT RECORD
*
***********************************************************************
* NOTES:                                                              *
* 1. NEXT PARAMETER IS FETCHED FROM COMMAND INTO TEXT FIELD 'PARM'    *
*    VIA CALL TO 'GETPARM'.  THERE IS NO RETURN IF THE SIZE OF THE    *
*    PARAMETER IS NOT WITHIN THE RANGE SPECIFIED BY THE 1ST AND 2ND   *
*    ARGUMENTS.                                                       *
* 2. ERRORS ARE LOGGED VIA CALL TO 'ERROR', WHICH DOES NOT RETURN.    *
***********************************************************************
*
***********************************************************************
* GET STATION NAME PARAMETER AND FIND STATION'S $.SYSNET MEMBER.      *
***********************************************************************
*
         CALL  GETPARM,1,8,(PNAME)         GET STATION NAME
         MOVE  STANAME,PARM,(8,BYTE)       ... AND SAVE IT
         CALL  FINDMEM,(STANAME),REC#      FIND MEMBER
         IF    (REC#,EQ,0)                 MEMBER NOT FOUND
           CALL  ERROR,12,(STANAME)        LOG STATION NOT DEFINED
         ENDIF
*
************************************************************************
* GET AND CHECK BIT NUMBER PARAMETER.                                  *
************************************************************************
*
         CALL  GETPARM,1,2,(PBIT)          GET BIT NUMBER
         CONVTD BITNUM,PARM,FORMAT=(2,0,I)  CONVERT TO BINARY
         IF    (S$CPBIT,NE,-1),OR,         NOT NUMERIC                 X
               (BITNUM,LT,0),OR,(BITNUM,GT,15)  OUT OF RANGE
           CALL  ERROR,9,(PBIT)            LOG INVALID PARAMETER
         ENDIF
*
************************************************************************
* GET AND CHECK BIT VALUE PARAMETER (OPTIONAL, DEFAULT IS 1).          *
************************************************************************
*
         MOVE  BITVAL,1                    SET DEFAULT VALUE
         CALL  GETPARM,0,1,(PVAL)          GET SPECIFIED VALUE
         IF    (PARM-1,NE,0,BYTE)          VALUE WAS SPECIFIED
           IF    (PARM,EQ,C'0',BYTE)       CHECK FOR VALUE = 0
             MOVE  BITVAL,0
           ELSE
             IF    (PARM,NE,C'1',BYTE)     CHECK FOR VALUE NOT = 1
               CALL  ERROR,9,(PVAL)        LOG INVALID PARAMETER
             ENDIF
           ENDIF
         ENDIF
*
```

Figure 29 (Part 1 of 5). S$CPBIT Listing

```
***********************************************************************
* SET MASK VALUE ACCORDING TO BIT NUMBER AND VALUE.                   *
***********************************************************************
*
         SHIFTL BITNUM,1                MULTIPLY BIT NUMBER BY 2
         MOVE  #1,BITNUM                SELECT CORRESPONDING MASK
         MOVE  MASK,(MASKS,#1)          ... FROM TABLE OF 'OR' MASKS
         IF    (BITVAL,EQ,0)            BIT IS TO BE TURNED OFF
           MOVE  #1,X'FFFF'             CHANGE 'OR' MASK
           SUB   #1,MASK,RESULT=MASK    ... TO 'AND' MASK
         ENDIF
*
***********************************************************************
* FIND STATION BLOCK.  IF STATION IS STARTED AND HAS A DISK QUEUE,    *
* ENQUEUE ON THE QCB IN ITS FILE CONTROL BLOCK.                       *
***********************************************************************
*
         MOVE  FCB@,0                   INDICATE NO FILE CTL BLOCK
         LOCATE ST,#2,STANAME           #2 ==> STATION BLOCK
         IF    (#2,NE,0)                STATION IS STARTED
           IF    ((Q$TYPE,#2),NE,+Q#VECT,BYTE),  IT'S NOT A VECTOR     X
               AND,((Q$DQA,#2),NE,0)            ... AND HAS DISK QUEUE
             MOVE  #1,(Q$DQA,#2)        GET ADDR OF FILE CONTROL
             MOVE  FCB@,#1              ... BLOCK AND SAVE IT
             ENQ   (DQQCB,#1)           ENQUEUE ON USE OF $.SYSNET
           ENDIF
         ENDIF
*
***********************************************************************
* UPDATE $.SYSNET MEMBER AND STATION BLOCK, IF IT EXISTS.             *
***********************************************************************
*
         READ  DS1,BUFF,1,REC#,ERROR=NETERR  READ MEMBER
         MOVEA #1,BUFF                  #1 ==> MEMBER RECORD
         IF    (BITVAL,EQ,1)            SET BIT ON
           IOR   (Q$STAT,#1),MASK       IN MEMBER RECORD
           IF    (#2,NE,0)
             IOR   (Q$STAT,#2),MASK     IN STATION BLOCK
           ENDIF
         ELSE                           SET BIT OFF
           AND   (Q$STAT,#1),MASK       IN MEMBER RECORD
           IF    (#2,NE,0)
             AND   (Q$STAT,#2),MASK     IN STATION BLOCK
           ENDIF
         ENDIF
         WRITE DS1,BUFF,1,REC#,ERROR=NETERR  WRITE MEMBER
         CALL  DEQNET                   RELEASE LOCK ON $.SYSNET
*
***********************************************************************
* LOG SUCCESSFUL COMPLETION AND STOP.                                 *
***********************************************************************
*
         MOVE  LOGMSG,STANAME,(8,BYTE)  MOVE STATION NAME TO MESSAGE
         SEND  LOG,1,LOGMSG,ID=C'UM'
STOP     EQU   *
         PROGSTOP  *,LOGMSG=NO,P1=ENDCODE
*
***********************************************************************
* ERROR ROUTINES.                                                     *
***********************************************************************
*
NETERR   EQU   *                        $.SYSNET DISK I/O ERROR
         MOVE  XCODE,DS1                XCODE = I/O RETURN CODE
         CALL  ERROR,3,(STANAME)        LOG ERROR
*
TEEXIT   EQU   *                        TASK ERROR EXIT
*                                       LOG HARDWARE STATUS INFO
         SEND  E,82,ID=C'CF',TYPE=X,XCODE=TEEHSA@*
         CALL  DEQNET                   RELEASE LOCK ON $.SYSNET
         GOTO  STOP                     TERMINATE EXECUTION
*
```

Figure 29 (Part 2 of 5). S$CPBIT Listing

```
***********************************************************************
* GETPARM: SUBROUTINE TO GET NEXT PARAMETER FROM COMMAND.             *
*          PARM-1 IS MINIMUM PARAMETER LENGTH.                        *
*          PARM-2 IS MAXIMUM PARAMETER LENGTH.                        *
*          PARM-3 IS ADDRESS OF PARAMETER NAME.                       *
*          IF PARAMETER LENGTH IS INVALID, EXIT IS TO 'STOP' VIA      *
*            CALL TO SUBROUTINE 'ERROR'.                              *
***********************************************************************
*
         SUBROUT GETPARM,PMIN,PMAX,PNAME@
         SHIFTL  PMAX,8,RESULT=PARM-2      SET MAXIMUM LENGTH
         DO      UNTIL,(PARM-1,NE,0,BYTE)  SKIP EXTRA BLANKS
           GET   FIELD,PARM,CPBFAD*,COMPARE=X'40'  GET PARAMETER
           IF      (S$CPBIT,EQ,4)          END OF BUFFER
             IF      (PMIN,EQ,0)           PARAMETER IS OPTIONAL
               RETURN
             ELSE
               CALL   ERROR,44,PNAME@      LOG PARAMETER OMITTED
             ENDIF
           ENDIF
         ENDDO
         IF      (S$CPBIT,EQ,2),OR,        PARAMETER TOO LONG         X
                 (PARM-1,LT,PMIN+1,BYTE)   ... OR TOO SHORT
           CALL   ERROR,9,PNAME@           LOG PARAMETER INVALID
         ENDIF
         RETURN
*
***********************************************************************
* ERROR: SUBROUTINE TO LOG AN ERROR MESSAGE, SET THE COMPLETION CODE, *
*        AND EXIT (DOES NOT RETURN TO CALLER).                        *
*        PARM-1 IS MSG NUMBER, ITS NEGATIVE VALUE IS COMPLETION CODE. *
*        PARM-2 IS ADDRESS OF TEXT TO BE INSERTED IN MESSAGE.         *
*        THE XCODE, IF ANY, IS SET BY THE CALLER.                     *
***********************************************************************
*
         SUBROUT  ERROR,ERR#,ERRTXT@
         CALL    DEQNET                    RELEASE LOCK ON $.SYSNET
         SEND    ERROR,ERR#*,ERRTXT@*,XCODE=XCODE*,ID=C'CP'
         MOVE    ENDCODE,0                 COMPLETION CODE IS NEGATIVE
         SUB     ENDCODE,ERR#              ... MESSAGE NUMBER
         GOTO    STOP                      TERMINATE EXECUTION
         RETURN
*
***********************************************************************
* DEQNET: SUBROUTINE TO RELEASE LOCK ON $.SYSNET  -- TO  DEQUEUE THE  *
*         QCB IN AN ACTIVE STATION'S FILE CONTROL BLOCK.              *
***********************************************************************
*
         SUBROUT DEQNET
         IF      (FCB@,NE,0)               WE MAY BE ENQUEUED
            MOVE    #2,FCB@                GET ADDR OF FILE CTL BLOCK
            IF      ((DQQCB+6,#2),EQ,S$CPBIT@) WE ARE ENQUEUED
              DEQ    (DQQCB,#2)            RELEASE
            ENDIF
         ENDIF
         RETURN
*
```

Figure 29 (Part 3 of 5). S$CPBIT Listing

```
************************************************************************
* FINDMEM: SUBROUTINE TO LOCATE A MEMBER IN DS1, A PARTITIONED DATA    *
*          SET, AND RETURN ITS RECORD NUMBER.                          *
*          PARM-1 IS ADDRESS OF MEMBER NAME.                           *
*          PARM-2 IS A WORD TO RECEIVE THE RECORD NUMBER; 0 MEANS THE  *
*                 MEMBER DOESN'T EXIST.                                *
************************************************************************
*
         SUBROUT  FINDMEM,FNAME@,FREC#
         MOVE  FSAVREGS,#1,2            SAVE CALLER'S REGISTERS
         MOVE  #2,FNAME@                #2 ==> MEMBER NAME
         MOVE  FREC#,0                  INDICATE MEMBER NOT FOUND
         MOVE  FDIR#,1                  DIRECTORY BEGINS IN RECORD 1
         MOVEA #1,BUFF                  #1 ==> INPUT BUFFER
         READ  DS1,(0,#1),1,FDIR#,ERROR=FEXIT    READ 1ST RECORD
         MOVE  FDEND,(F$DNXTE,#1)       SAVE # OF NEXT AVAIL ENTRY
         ADD   #1,+F$EL                 POINT TO FIRST MEMBER ENTRY
         MOVE  FMEM#,1                  INIT MEMBER ENTRY NUMBER
         MOVE  FCOUNT,+F$E#             SET LOOP COUNT, WHICH DOES
         SUB   FCOUNT,1                 ... NOT INCLUDE FIRST ENTRY
FSEARCH  EQU   *
         DO    FCOUNT,TIMES             EXAMINE EACH ENTRY IN RECD
           IF     (FMEM#,EQ,FDEND),GOTO,FEXIT  DIRECTORY END,
*                                              ...MEMBER NOT FOUND
           IF     ((F$MNAME,#1),EQ,(0,#2),8)   FOUND MEMBER
             IF     ((F$MMCODE,#1),NE,+F$MDEL) IT'S NOT DELETED
               MOVE  FREC#,(F$MREC1,#1)    SET 1ST RECORD NUMBER
FEXIT          EQU   *
               MOVE  #1,FSAVREGS,2         RESTORE REGISTERS
               RETURN                      EXIT
             ENDIF
           ENDIF
           ADD   #1,+F$EL               POINT TO NEXT ENTRY
           ADD   FMEM#,1                INCREMENT ENTRY NUMBER
         ENDDO
         ADD   FDIR#,1                  INCREMENT RECORD NUMBER
         MOVEA #1,BUFF                  RESET BUFFER ADDRESS
         MOVE  FCOUNT,+F$E#             RESET LOOP COUNT
         READ  DS1,(0,#1),1,FDIR#,ERROR=FEXIT    GET NEXT RECORD
         GOTO  FSEARCH
*
FSAVREGS DATA  2F'0'           REGISTER SAVE AREA
FDIR#    DATA  F'0'            DIRECTORY RECORD NUMBER
FDEND    DATA  F'0'            NUMBER OF NEXT AVAIL DIR ENTRY
FMEM#    DATA  F'0'            MEMBER ENTRY NUMBER
FCOUNT   DATA  F'0'            LOOP CONTROL
*
************************************************************************
* PARTITIONED DATA SET DIRECTORY DEFINITIONS.                          *
* THERE ARE 16 ENTRIES PER DIRECTORY RECORD, EACH 16 BYTES.            *
* 1ST ENTRY DESCRIBES DATA SET; EACH SUBSEQUENT ENTRY DESCRIBES A      *
*  MEMBER OF DATA SET.                                                 *
************************************************************************
*
F$D      EQU   0               DATA SET ENTRY
F$DNXTM  EQU   F$D             NEXT AVAILABLE RECORD NUM FOR MEMBER
F$DSIZE  EQU   F$DNXTM+2       TOTAL SIZE OF DATA SET IN RECORDS
F$DNXTE  EQU   F$DSIZE+2       NEXT AVAILABLE DIR ENTRY (ORIGIN 0)
F$DTOTE  EQU   F$DNXTE+2       TOTAL MEMBER ENTRIES (USED OR NOT)
*                              LAST 8 BYTES ARE NOT USED
*
F$M      EQU   0               MEMBER ENTRY
F$MNAME  EQU   F$M             MEMBER NAME
F$MREC1  EQU   F$MNAME+8       1ST RECD NUM (RELATIVE TO DS START)
F$M#REC  EQU   F$MREC1+2       NUMBER OF RECORDS IN MEMBER
F$MMCODE EQU   F$M#REC+2       MEMBER CODE:
F$MDEL   EQU   -1                MEMBER IS DELETED
F$MUCODE EQU   F$MMCODE+2      USER CODE
F$EL     EQU   F$MUCODE+2      LENGTH OF DIRECTORY ENTRY
*
F$E#     EQU   16              NUMBER OF ENTRIES PER RECORD
```

Figure 29 (Part 4 of 5). S$CPBIT Listing

```
*
***********************************************************************
* CONSTANTS                                                           *
***********************************************************************
*
LOGMSG   TEXT  'XXXXXXXX STATUS MODIFIED'  INFORMATION MESSAGE
S$CPBIT@ DATA  A(S$CPBIT)                    TASK ADDRESS
*                                            PARAMETER NAMES FOR ERROR MSGS
PNAME    TEXT  'STATION NAME'
PBIT     TEXT  'BIT NUMBER'
PVAL     TEXT  'BIT VALUE'
*                                            MASKS FOR OR-ING BITS 0 - 15
MASKS    DATA  X'8000',X'4000',X'2000',X'1000'
         DATA  X'0800',X'0400',X'0200',X'0100'
         DATA  X'0080',X'0040',X'0020',X'0010'
         DATA  X'0008',X'0004',X'0002',X'0001'
TEECB    EQU   *                             TASK ERROR EXIT CONTROL BLOCK
TEECTL   DATA  X'0002'                       NUMBER OF FOLLOWING WORDS
TEEXIT@  DATA  A(TEEXIT)                     ADDR OF TASK ERROR EXIT ROUTINE
TEEHSA@  DATA  A(TEEHSA)                     ADDR OF HARDWARE STATUS AREA
*
***********************************************************************
* WORK AREAS                                                          *
***********************************************************************
*
CPBFAD   DATA  F'0'                          ADDR OF CP INPUT BUFFER
BUFF     DATA  128X'0000'                    DISK BUFFER
TEEHSA   EQU   BUFF                          HARDWARE STATUS AREA
REC#     DATA  F'0'                          MEMBER RECORD NUMBER
FCB@     DATA  F'0'                          ADDR OF FILE CONTROL BLOCK
XCODE    DATA  F'0'                          LOG MESSAGE XCODE
BITNUM   DATA  F'0'                          BIT NUMBER
BITVAL   DATA  F'0'                          BIT VALUE
MASK     DATA  F'0'                          MASK FOR MODIFYING STATUS
STANAME  TEXT  LENGTH=8                      STATION NAME
PARM     TEXT  LENGTH=8                      PARAMETER INPUT AREA
*
         ENDPROG
         END
```

Figure 29 (Part 5 of 5). S$CPBIT Listing

Creating an I/O Control Program

You can extend the Communications Facility to support a new device or a new Series/1 feature by writing an I/O control program (IOCP). I/O control programs provide the device-dependent functions necessary to control input to and output from the device.

Your I/O control program must accept messages destined for the device from the Communications Facility, process them, and forward them to the device. It must also accept messages from the device and pass them along to the Communications Facility for delivery to their ultimate destination. The Communications Facility treats your I/O control program as a user program.

Overview of an I/O Control Program

The primary functions of an I/O control program are:

- Receive messages from stations that represent the physical devices, using Communications Facility language extension instructions.
- Reformat those messages as required.
- Send messages to the physical devices, using device-specific routines.
- Receive messages from the physical devices, using device-specific routines.
- Reformat those messages as required.
- Send those messages to their destination stations, using Communications Facility language extension instructions.

Additionally, the I/O control program must handle control input from the Communications Facility, such as instructions to halt itself or one of its devices. The I/O control programs supplied with the Communications Facility handle such functions in a separate task. A main task receives control information from the Communications Facility and oversees operation of one or more subtasks that service message traffic.

Designing Your I/O Control Program

When designing your I/O control program, your first consideration is what functions you intend it to add to the Communications Facility system. For example, your I/O control program might manage a specific device as if it were a 3270 terminal. Your program would then have to provide all 3270 data stream support, mapping device functions to 3270 control keys and allowing the device to communicate with existing Communications Facility 3270 I/O control programs, such as 3270 emulation.

As another example, your program might provide a new way to connect Communications Facility nodes over a communication line or network not directly supported by the Communications Facility. As a third example, you might want to allow your Communications Facility user programs to communicate with a special type of device, without including any special data stream conversions.

Another consideration is whether you're going to support one device or multiple devices. The I/O control programs that are supplied with the Communications

Facility support multiple devices or lines simultaneously; for example, the 3270 control IOCP can support several 3270 terminals at once. You must decide whether you need such flexibility, or whether supporting a single device will be adequate. Supporting multiple devices may introduce the need for multiple tasks and reentrant code, whereas single-device IOCPs can be more straightforward.

You must also, of course, consider what hardware and what programming techniques will be necessary to control the device you intend to support.

Finally, you must decide how you will represent your device to the Communications Facility. You may make use of existing station types (such as terminal, line, or device), add new station types, or build special stations when your I/O control program is loaded.

Creating Your Stations

Because all messages and commands in the Communications Facility flow to and from stations, you have to create stations to represent the devices your IOCP supports. This section explains three ways of creating stations.

The first, and recommended, method is to build special station blocks directly from the I/O control program main task. When the program is loaded, it issues LOCATE instructions with OPTION=CREATE to build station blocks to your specifications in S$POOL. Having built the station blocks, you can use MOVE instructions to fill in their fields. You can examine S$CFEQU or look in the *Debugging Guide* for a description of the fields of station blocks.

A second possibility is to use the station types and subtypes associated with a Communications Facility IOCP that your installation doesn't use. For example, if your IOCP controls some kind of device on a BSC line, and you know that your installation will never use the 3270 control IOCP ($.IO0AC0), you could name your IOCP $.IO0AC0 and use line and terminal stations as defined by that program. Then you could use the standard Communications Facility Start, Define, Help, Modify, Stop, and Halt commands to manage your stations.

To use this option, your program must be coded to use the stations just as the original IOCP does; your station blocks must be of exactly the same size and format as those of the original IOCP.

If you choose this option and later your installation elects to use the IOCP you have replaced, you will have to rework your IOCP extensively.

A third method is to define new station types and modify the Start, Define, Modify, Stop, and Halt commands to handle them. The main drawback of this method is that it involves modifying Communications Facility modules, and so may cause serious maintenance and service problems. If you do choose this method, select the modules you need to modify from the "Module Descriptions" list in the *Debugging Guide.*

Coding the Main Task

The main task accepts commands for the individual devices controlled by the IOCP.

If you have modified the CP command processor or if you're using the station types of a Communications Facility IOCP, your main task will receive control commands directed to your stations. In that case, you can refer to the listing of the main task of one of the other IOCPs for guidance. S$IO0AC0, 3270 control, is a typical example.

If you're using the LOCATE OPTION=CREATE technique, you'll have to build in code to receive and interpret commands.

When your IOCP processes a Start command, it may start a subtask to handle the started device. If the device is attached directly, use a subtask for each device. If several devices are attached over a line, use one subtask per line and have that subtask control all the devices. The ACTIVATE T instruction is a handy way to start an IOCP subtask.

Coding Subtasks

The subtask processes message traffic between your device and the Communications Facility. Its main functions are those listed earlier under "Overview of an I/O Control Program" on page 165.

You must devise a method of mapping each device to its associated station block. You might choose device address (as used in the 3101 and 4978 IOCPs), poll and select sequence (as used in 3270 emulation and control), or some other method. Whatever method you choose, your subtask then uses it to select the station block to be used for message traffic to a device. The subtask moves messages from a specific station block to a specific device, and vice versa.

The subtask must also accept control input from the main task, and detach itself when requested. The usual method for this is to monitor the Q#ACTIVE bit in field Q$STAT of the station block. When Q#ACTIVE is off, the subtask performs whatever operations are necessary to prepare for termination, and then terminates. Several bits of Q$STAT are left as device-dependent bits; your main task and subtasks can use those bits for their own purposes.

Reentrant Coding Considerations

If your subtask code is reentrant, you can save storage by using the same subtask code for all your devices.

Note that some EDX instructions and techniques are not reentrant; you can't use them in a reentrant subtask. Examples are:

- The P*n* feature of EDL instructions
- The SUBROUT, CALL, and RETURN instructions
- The DO *count*,TIMES form of the DO instructions.

Availability of only two registers can make reentrant coding difficult. The #T, #L, and indirect (*) features of Communications Facility instructions alleviate the problem somewhat. If you locate your EDX task control block (TCB) and your station working storage next to your station block, you can use a single register to refer to all of them. If you also use #T, you can often free both EDX registers temporarily.

You might, alternatively, locate the TCB and working storage in a workspace acquired by each task. You can use the PUT TCB instruction to build the TCB. This method requires the use of two registers to refer to the station block and working storage at the same time. Note that space used in the station block is in the mapped area of EDX; increasing its size will reduce the amount of storage available for programs in all Communications Facility partitions.

Data Stream Considerations—3270 IOCPs

If your I/O control program manages a device as if it were a 3270 terminal, it must build messages according to the expectations of the other 3270-type I/O control programs.

Messages passing between such programs use two format modes: basic mode and record mode. Basic mode messages are formatted in normal BSC 3270 data stream format, including BSC control characters before and after the actual 3270 screen. Record mode messages are sent and received without the special BSC and 3270 control characters.

Basic Mode Messages

To coexist with Communications Facility 3270 programs, your program must be coded to accept and send basic mode messages.

The format of a typical basic mode message as it flows from a program to a terminal is:

STX	ESC	*cmd*	*wcc*	orders and data	ETX

The format of a typical basic mode message as it flows from a terminal to a program is:

STX	*cuda*	AID	*cursor*	orders and data	ETX

STX
is X'02', the BSC start text control character.

ESC
is X'27', the BSC escape control character.

cmd
is the 3270 remote command code, such as write or erase/write. Its values are defined in the *3270 Description and Programmer's Guide.*

wcc
is the 3270 write control character. Its values are defined in the *3270 Description and Programmer's Guide.*

ETX
is X'03', the BSC end text control character. It may or may not appear in a basic mode message.

cuda
is the BSC control unit/device address field. Within the Communications Facility, these values are meaningless; they are often set to blanks, X'4040'.

AID
is the 3270 attention identifier. Its values are defined in the *3270 Description and Programmer's Guide.*

cursor
is the position of the cursor on the 3270 screen when the 3270 attention-type key was pressed. It is expressed in 3270 position address values, as described in the *3270 Description and Programmer's Guide.*

Basic mode messages may also begin with a BSC DLE/STX sequence (X'1002') instead of STX. DLE/STX indicates that the message should be transmitted in transparent mode—that is, accepting any bit sequence found in the data stream without changing it or interpreting it as a BSC control character.

Record Mode Messages

Record mode is normally used for transmitting, over a 3270 line, messages built without standard 3270 control sequences. Another limited use is for connecting two 3270-type terminals so they can send messages to one another directly without the intervention of an application program.

IOCPs support record mode by adding control characters before sending a message over a line or to a device, and by removing the characters before giving a message to the message dispatcher.

Four kinds of record mode processing can occur:

- An IOCP connecting a host to the Communications Facility receives a message for the host.
- An IOCP connecting a device to the Communications Facility receives a message for its device.
- An IOCP connecting a host to the Communications Facility receives a message from the host.
- An IOCP connecting a device to the Communications Facility receives a message from its device.

I/O control programs receiving a message from the Communications Facility assume that the message is in record mode when it is received without STX (X'02') or DLE STX (X'1002') as the starting character sequence.

In the case of a message being sent from the Communications Facility to a device, a default 3270 remote command code of X'F5' (erase/write) and a write control character of X'F8' (80-character line) are prefixed to the message, along with whatever device control is necessary.

In the case of a message being sent from the Communications Facility to a host, an attention ID value of X'7D' (ENTER key) and a cursor address of X'4040' (screen position 0) are added to the message. See the actual source code of the Communications Facility I/O control programs to become familiar with this technique.

Your IOCP may put messages received from a 3270 line for the Communications Facility into record mode by removing control characters from the message before sending it. Check the record mode bit of the Q$STAT field of the station block to determine whether or not to use record mode.

To put messages received from the host into record mode, follow these steps, as used by the Communications Facility 3270 IOCPs:

- Remove the STX (X'02') and ESC (X'27') characters from the message (for BSC) or don't add it (for other line protocols).
- Remove remote command code and write control characters.
- If the next character is a 3270 set buffer address order (X'11'), remove three characters representing the order and the address operand.
- If the next character is a 3270 start field order (X'1D'), remove two characters representing the order and the field attribute byte.
- Remove the ETX (X'03') from the end of the message (for BSC) or don't add it (for other line protocols).

Messages received from a device are processed similarly; the difference is in the format of the beginning of such a message:

- Remove the STX (X'02') and 3270 control unit and device address fields from the message (for BSC) or don't add them (for other line protocols).
- Remove the 3270 attention identifier and 2-byte cursor position address.
- If the next character is a 3270 set buffer address order (X'11'), remove three characters representing the order and the address operand.
- Remove the ETX (X'03') from the end of the message (for BSC) or don't add it (for other line protocols).

In neither case is the data stream inside the message modified; it is assumed that the originator of the message has set up the message content beyond the fields that have been removed.

Examine the source code of the various 3270 IOCPs for detailed information about the record mode processing each one does.

Data Stream Considerations—IOCPs that Connect Nodes

Your IOCP may connect Communications Facility nodes, as do the Series/1-to-Series/1 IOCP ($.IO0A10), the X.25 IOCP ($.IO0AB8), or the Local Communications Controller IOCP ($.IO0AB0). You must receive messages from the Communications Facility that have Communications Facility message headers; transmit messages between nodes with message headers intact; and supply the appropriate message headers when passing messages to the Communications Facility in the other node. To pick up and send the headers, use the HEADER= operand of the SEND or RECEIVE instruction.

Because the headers may contain any random bit pattern, data transmission must be transparent in this type of program.

If you're writing a program of this type, you can use $.IO0AB0 (the Local Communications Controller IOCP) or $.IO0AB8 (the X.25 IOCP) as examples. $.IO0A10 (the Series/1-to-Series/1 IOCP) isn't a suitable example, because it has many dependencies on Communications Facilities internals that might change.

Example I/O Control Program Listing

This section describes a BSC point-to-point IOCP in detail. The point-to-point IOCP need only issue a RECEIVE instruction from the line station and write data to the line continually until it receives no message. At this point, the program reads the line, sends any data to the Communications Facility, and issues another RECEIVE instruction.

Figure 30 on page 172 shows an example IOCP. The sample is *not* an operational program. It shows only enough logic to illustrate its operation.

Statement 200 defines an EDX attention list containing the commands used to control the IOCP.

Statement 300 copies in the Communications Facility equate list for use by the IOCP.

Statements 500 to 1000 define the BSC IOCB in the work area allocated at the end of the line station block. These locations are accessed as displacement values from register #1.

Statement 1200 locates the station block for the IOCP. The address is returned in register 1.

Statement 1300 turns on the station active bit for the IOCP's station.

Statement 1500 waits for one of the attention routines to post CMDECB, indicating that a command has been received.

Statements 1600 to 1900 perform the ISTOP command. The program sends an END message to the log, and issues a PROGSTOP.

Statement 2000 checks for an ISTART command. If none exists, it returns to statement 1500 to await a command.

Statement 2100 tries to locate the line station block. If the locate is successful, the address is returned in register 2; if not, register 2 is cleared and the program goes to the instruction labeled GETL.

Statement 2200 sets the active field in the status field of the station block.

```
IOSAMPLE PROGRAM START,450                                             00000100
         ATTNLIST (ISTOP,SP,ISTART,ST)                                 00000200
         COPY  S$CFEQU                                                 00000300
*                                                                      00000400
BSIOCB   EQU   Q$END               BSC IOCB                            00000500
BSIODA   EQU   BSIOCB+2            BSC DEVICE ADDRESS                  00000600
BSBUF1   EQU   BSIODA+2            BSC BUFFER 1                        00000700
BSL1     EQU   BSBUF1+2            BSC BUFFER 1 LENGTH                 00000800
BSBUF2   EQU   BSL1+2              BSC BUFFER 2                        00000900
BSL2     EQU   BSBUF2+2            BSC BUFFER 2 LENGTH                 00001000
*                                                                      00001100
START    LOCATE LUNAME,#1,OPTION=CREATE  BUILD IOCP STATION            00001200
         IOR   (Q$STAT,#1),+Q#ACTIVE SET LU ACTIVE                     00001300
*                                                                      00001400
GETL     WAIT  CMDECB              WAIT FOR COMMAND TO PROCESS         00001500
         IF    (CMDTYPE,EQ,+IP),THEN  ISTOP COMMAND                    00001600
           SEND  LOG,21,XCODE=IOSAMPLE*,TYPE=I,ID=C'IO'                00001700
           PROGSTOP                                                    00001800
         ENDIF                                                         00001900
         IF    (CMDTYPE,NE,+IS),GOTO,GETL  INVALID COMMAND             00002000
         LOCATE LU,#2,LUNAME,EXIT=GETL BUILD LINE STATION              00002100
         IOR   (Q$STAT,#2),+Q#ACTIVE SET STATION ACTIVE                00002200
         ACTIVATE TASK,(Q$TCB,#2),STARTL  START LINE TASK              00002300
         GOTO  GETL                                                    00002400
*                                                                      00002500
STARTL   GET   ADDRESS,#2,#LINE                                        00002600
         GET   BUFFER,(BSBUF1,#2)*,(Q$BFSZ,#2),POOL=IOPOOL             00002700
         MOVE  (BSL1,#2),(Q$BFSZ,#2)                                   00002800
         MOVE  (BSL2,#2),(Q$BFSZ,#2)                                   00002900
         AND   (Q$NAU,#2),X'00FF',RESULT=(BSIODA,#2)                   00003000
*                                                                      00003100
OUTPUT   RECEIVE MESSAGE,(BSBUF1,#2)*,(Q$NAME,#2),WAIT=NO,            CC00003200
               EXIT=INPUT                                              00003300
         BSCWRITE I,(BSCIOCB,#2)                                       00003400
         GOTO  OUTPUT              GET NEXT MESSAGE                    00003500
*                                                                      00003600
INPUT    BSCWRITE E,(BSIOCB,#2)                                        00003700
         BSCREAD I,(BSIOCB,#2),END=OUTPUT,ERROR=OUTPUT                 00003800
         SEND  MESSAGE,,(BSBUF1,#2)*,ORIGIN=(Q$NAME,#2)                00003900
         GOTO  INPUT                                                   00004000
*                                                                      00004100
ISP      EQU   *                   ISTOP COMMAND ENTERED               00004200
         POST  CMDECB,+IP          PROCESS AN ISTOP COMMAND            00004300
         ENDATTN                                                       00004400
*                                                                      00004500
IST      EQU   *                   ISTART COMMAND ENTERED              00004600
         POST  CMDECB,+IS          PROCESS AN ISTART COMMAND           00004700
         ENDATTN                                                       00004800
*                                                                      00004900
IOPOOL   DEFINE POOL,SIZE=2000     BUFFER POOL                         00005000
*                                                                      00005100
LUNAME   TEXT  'PTPTLINE'          THE ONE AND ONLY STATION            00005200
CMDECB   ECB   0                   ECB POSTED WHEN COMMAND IS ENTRD    00005300
CMDTYPE  EQU   CMDECB              COMMAND TYPE FROM ISTART/ISTOP      00005400
IS       EQU   1                   ISTART COMMAND TYPE                 00005500
IP       EQU   2                   ISTOP  COMMAND TYPE                 00005600
         ENDPROG                                                       00005700
         END                                                           00005800
```

Figure 30. Sample IOCP Listing

If the command is ISTART and the line station block is located, statement 2300 starts the line subtask. The task control block is located at Q$TCB in the line station block, and the subtask is to start execution at STARTL. The program continues at statement 2400 and waits for another message at statement 1500.

The next execution occurs when the subtask begins at STARTL in statement 2600. The subtask gets the address of the line station block in register 2. The equates in C$CFEQU that begin with Q$ can then be used.

At statement 2700, the program gets buffer space from the pool of workspace in the program. The pool's size was set during $.CONFIG execution and is stored as an integer in Q$BFSIZ in the line station block. In this example, the buffer is kept and never freed.

Statements 2800 to 3000 set up the IOCB needed by the BSC access method.

Statement 3200 starts the I/O processing. The program issues a RECEIVE instruction for a message queued to the station named at Q$NAME in the line station block. It moves the message into the buffer whose address is at BSBUF1. If no message is received, execution continues at INPUT (statement 3700). If data is received, statement 3400 writes the data to the BSC line, using the IOCB in the line station block work area.

After the BSC write operation, the program continues at statement 3200, where eventually the RECEIVE exits to INPUT. At INPUT (statement 3700), the program writes an EOT and reads data with a BSCREAD in statement 3800.

After the read is complete, execution continues at OUTPUT in statement 3200 if an EOT is received or at statement 3900 if data is received.

The SEND instruction sends a message to the message dispatcher. The message is sent from the buffer specified to the BSC IOCB, and the origin of the message is specified as the Q$NAME of the line station block. After the SEND is complete, execution continues at OUTPUT.

Statements 4200 through 4400 are the attention processing routine that handles the attention command ISTOP. It posts the CMDECB with a code indicating the type of command. Statements 4600 through 4800 do the same thing for the ISTART command. When CMDECB is posted, the IOCP waiting at statement 1500 begins execution.

The line buffers are acquired from a workspace pool defined at statement 5000.

Coding Communications Facility Instructions

This chapter gives the details of the format, operands, and return codes of each Communications Facility language extension instruction except those that are used to access the Local Communications Controller. The Local Communications Controller instructions are intended for internal use only and are documented in the *Debugging Guide*.

First, here's an explanation of the syntax notation that's used to describe the instructions in this chapter.

Syntax Notation

Each instruction format in this chapter appears in a box with three columns:

Name	Operation	Operand

Name Field. This field contains a symbolic label of up to 8 characters.

Operation Field. This field contains the instruction name.

Operand Field. This field contains the operands associated with the instruction.

The following conventions are used within the format descriptions:

- Words in **BOLD CAPITAL** letters must be coded exactly as shown. Commas, parentheses, and equal signs must also be coded exactly as shown.
- Values in **BOLD UNDERSCORED** letters are defaults.
- Words in *italics* are symbols for which you must substitute actual values.
- Brackets ([ ]) indicate that the operand is optional.
- Braces ({ }) indicate a group of mutually exclusive operands or values, of which you can code only one.
- A vertical bar (|) separates the mutually exclusive items within braces.

Special Symbolic Addresses

To facilitate using EDL language for manipulation of data in system control blocks, two special symbolic addresses are allowed in the language extension instructions:

- #T—refers to the beginning of the current task control block ($TCBCO or $TCBECB).
- #L—refers to the beginning of the station block associated with the current TCB. Either the TCB must be within the station block, or the word following the TCB must contain the address of the station block.

Instruction Format

The syntax of the language extensions is very similar to that of the other EDL instructions, except that most operands in the language extensions can be indexed and indirect. For information about EDL syntax rules, see the *EDL Reference* manual.

The general format of a Communications Facility language extension instruction is:

label **VERB MODIFIER,***pos1,pos2,*. . .,**KEY1**=*value1*,**KEY2**=*value2*,. . .,**P1**=*name1*,**P2**=*name2*,. . .

where:

label
is the label to be given to the first word of the generated instruction. It is required only if other instructions will refer to this instruction.

VERB
is the name of the general operation to be performed (such as DEFINE, SEND, or GET).

MODIFIER
is an additional word (such as BUFFER, MESSAGE, or STORAGE) that further defines what the instruction is to do. Three instructions, CFTERM, LCC, and MOV, have no modifier.

The modifier names as given in this chapter are the minimum names you can code to specify the instructions. You can code additional letters beyond those required if you want. For example, GET B is the required form of the instruction that gets a buffer, but you can code GET BUFFER (or, for that matter, GET BANANA) if you want. Be careful to code the modifiers accurately. GET BIELD, for example, would be interpreted as a GET BUFFER instruction, not a GET FIELD instruction.

In some cases, for compatibility with earlier versions of the Communications Facility, different modifiers have the same meaning. For example, LOCATE ST (locate station) has the same meaning as LOCATE LU (locate logical unit) and LOCATE QN (locate queue name). In those cases, all the allowable modifiers are shown in the instruction syntax.

pos1, pos2,. . .
are positional operands, which must be coded in the position shown.

KEY1, KEY2,. . .
are keyword operands, for which you supply values.

P1, P2,. . .
allow you to give names to operands, as in other EDL instructions.

A space is required after the verb. No spaces are permitted between the modifier and the first operand, or between operands.

Operand Formats

The description of each operand in this chapter includes a notation of its valid syntax. The terms used to describe the syntax are:

#R
: EDX software registers, #1 or #2.

#A
: The language extension symbolic addresses #T and #L.

disp
: A displacement value to be added to an index register value to obtain an effective address. A displacement is normally represented by an equated value or a self-defining term.

baddr
: A base address value to be added to an index register value to obtain an effective address. This is normally the label of a storage location.

term
: A self-defining term of 2 bytes or less. Self-defining terms are decimal constants such as the bufferpool size 4000, hexadecimal constants such as the polling address X'C240', or EBCDIC constants such as the characters C'NAME'.

literal
: A specific value, specified in the syntax of the instruction that requires its use. For example, a literal may be YES, NO, BUFFER, or CREATE. A literal may also be alphameric such as the volume name EDX002 or hexadecimal such as the device address A9.

'string'
: Any valid EBCDIC character string, enclosed in single quotes. The maximum length of the string, if applicable, is given in the operand description. Station names can be 1 to 8 characters; text strings can be 1 to 254 characters.

label
: Either a symbolic name to represent the generated instruction, or a nonindexable label of another storage location (used in the EXIT operand of several instructions).

*
: An indicator of an indirect reference.

location
: An address, which you can specify in any of these ways:

- *label*. The label defines the effective address.
- *label**. The storage location defined by the label contains the effective address.
- #1 or #2. The EDX software register is the effective address.
- #R*. The EDX software register contains the effective address.

- #A*. The storage location defined by #A contains the the effective address.

- (*disp*,#R) or (*baddr*,#R). The displacement or base address plus the value in the EDX software register is the effective address.

- (*disp*,#A). The storage location defined by #A plus the displacement is the effective address.

- (*disp*,#R)* or (*baddr*,#R)*. The storage location defined by the displacement or base address plus the value in the EDX software register contains the effective address.

- (*disp*,#A)*. The storage location defined by #A plus the displacement contains the effective address.

integer
: An integer value, which you can specify in any of these ways:

- *term*. The *term*, which may be a self-defining term or a value defined by an EQU statement, is the integer.

- *label**. The storage location defined by the label contains the integer.

- #R*. The EDX software register contains the integer.

- #A*. The storage location defined by #A contains the integer.

- (*disp*,#R)* or (*baddr*,#R)*. The storage location defined by the displacement or base register plus the value in the EDX software register contains the integer.

- (*disp*,#A)*. The storage location defined by #A plus the displacement contains the integer.

Macro Assembler Considerations

This section describes considerations that apply when you use the EDX Macro Assembler (5719-ASA) to generate language extension instructions.

All labels, terms, expressions, and *baddr* operands are subject to limitations imposed by the assembler. The maximum length of any of these items is determined by the maximum length of an operand in the SETC instruction. For the EDX macro assembler, the maximum is 64 characters; this limitation is subject to change, and you should check the documentation of the assembler you're using.

If you want to use an expression larger than the size shown, you can break it up into several smaller expressions using equates, finally resolving to a term or expression that is small enough.

ACTIVATE T—Activate or Deactivate a Task

The ACTIVATE T instruction attaches and chains, unchains, or detaches and unchains a task.

ACTIVATE T Format

[*label*]	**ACTIVATE T**	[,*tcbaddr*] [,*start*] **[,OPTION={UNCHAIN | DETACH}]** **[,EXIT=***label***]** **[,ERRXIT=***label***]** **[,P1=***name1***]** **[,P2=***name2***]**

ACTIVATE T Operands

Pn	Operand	Syntax	Description
P1	*tcbaddr*	location	The address of the TCB for the task to be activated or unchained. This operand is not valid for OPTION=DETACH.
P2	*start*	location	The starting address of the task to be attached. This operand is not valid for OPTION=UNCHAIN or OPTION=DETACH.
	OPTION=	literal	UNCHAIN causes the named task to be taken out of the EDX task chain; DETACH causes the issuing task to be detached and removed from the EDX task chain.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.
	ERRXIT=	label	The label of a task error exit control block. See "Using Task Error Exits" on page 29 for more information.

ACTIVATE T Return Codes

-1	Successful.
1	If OPTION= was not specified, the TCB is invalid. If OPTION= was specified, the detach or unchain was unsuccessful.
2	If OPTION= was not specified, the task is already activated. If OPTION= was specified, the task is already unchained.

ACTIVATE T Examples

```
1 ACTIVATE TASK,(Q$TCB,#2),START,EXIT=ERROR

2 ACTIVATE T,(Q$TCB,#2),OPTION=UNCHAIN

3 ACTIVATE T,OPTION=DETACH
```

Example 1 activates the task whose TCB is at (Q$TCB,#2). The task begins execution at location START. If a positive return code is returned, control is to go to the instruction labeled ERROR.

Example 2 unchains the task whose TCB is at (Q$TCB,#2).

Example 3 unchains and detaches the issuing task.

CFTERM—Define Non-EDX 3101 or 7485 on Multifunction Attachment

CFTERM defines a 3101 or 7485 terminal, on a multifunction attachment, that is intended for Communications Facility usage only; it is not known to the EDX supervisor. One or more of these instructions are used in conjunction with the ADAPTER statement that defines the multifunction attachment.

CFTERM is used only during EDX system generation.

CFTERM Format

label	**CFTERM**	**ADDRESS=***addr* **,LMODE={RS422 | LOCAL | SWITCHED | PTTOPT}**

CFTERM Operands

Operand	**Syntax**	**Description**
label	label	The name that identifies this terminal in the ADAPTER statement.
ADDRESS=	literal	The hexadecimal address of the multifunction attachment port to which the 3101 is connected.
LMODE=	literal	The type of connection.
	RS422	For a terminal directly attached to any port of the multifunction attachment. For a 7485, you must define LMODE=RS422.
	LOCAL	For a terminal directly attached. The terminal must be attached on the base address only.
	SWITCHED	For a point-to-point switched connection. The terminal must be attached on the base address only.
	PTTOPT	For a point-to-point nonswitched connection. The terminal must be attached on the base address only.

CFTERM Examples

```
MFA3101 ADAPTER ADDRESS=58,TYPE=MFA,              CC
                DEVICES=(T3101A,T3101B,T3101C), CC
                END=YES
T3101A  CFTERM  ADDRESS=58,LMODE=RS422
T3101B  CFTERM  ADDRESS=59,LMODE=RS422
T3101C  CFTERM  ADDRESS=5A,LMODE=RS422
```

These statements define a multifunction attachment with 3101 terminals attached to three ports.

DEFINE BRB—Define a Buffer Reference Block

Instead of using the DEFINE BUFFERPOOL or DEFINE WORK instruction to create the system storage pool, you may use the DEFINE BRB instruction to create a control block, called a *buffer reference block*, that defines the beginning of S$POOL. Modules that will not be needed in storage after EDX system initialization may be located in the area following the buffer reference block. After initialization, this area becomes S$POOL.

DEFINE BRB is used only during EDX system generation.

DEFINE BRB Format

[*label*]	**DEFINE BRB**	**,SIZE=***size*

DEFINE BRB Operands

Operand	Syntax	Description
label	label	The label to be assigned to the buffer reference block. If you code a label, it must be S$POOL.
SIZE=	term	The decimal size, in bytes, of the storage area to be allocated (the pool size plus 8).

DEFINE BRB Example

```
DEFINE BRB,SIZE=3816
```

creates this buffer reference block:

Address	Contents	
1100[4]	1108	*Pointer to first word after BRB*
1102	0000 0000	*2 full words of 0*
1106	0EE8	SIZE
1108		*First word after BRB*

4 Example address only

DEFINE BUFFER—Define a Buffer

DEFINE BUFFER creates a Communications Facility buffer in your program.

DEFINE BUFFER Format

label	**DEFINE BUFFER**	**,SIZE=***size* **[,DATA=**{*byte* | **X'40'**}**]** **[,P1=***name1***]**[5] **[,P2=***name2***]** **[,P3=***name3***]** **[,P4=***name4***]** **[,P5=***name5***]**

DEFINE BUFFER Operands

Operand	**Syntax**	**Description**
label	label	The label to be used to refer to the beginning of the data in the buffer.
SIZE=	term	The size, in bytes, of the buffer. The maximum size is 32767. Note that the size does not include the 10-byte buffer header. If the size you specify is 0 or negative, a buffer header with the 0 or negative value in its size field is created.
DATA=	term	A self-defining 1-byte constant to be used to initialize the buffer. For example, you might choose C'0' or X'00'.

DEFINE BUFFER Return Codes

None.

DEFINE BUFFER Examples

```
1 BUF1 DEFINE BUFFER,SIZE=200,P2=BCOUNT1

2 BUF2 DEFINE BUFFER,SIZE=80,DATA=X'F0'
```

Example 1 defines a 200-byte Communications Facility buffer, filled with EBCDIC blanks. You can refer to the count field in the buffer's header as BCOUNT1.

Example 2 defines an 80-byte Communications Facility buffer, filled with the character X'F0'.

5 The P*n* operands refer to the five words of the buffer header, described under "Getting a Buffer from Your Program" on page 7.

DEFINE BUFFERPOOL—Define a Workspace Pool

This instruction defines a pool of workspace within a program. When coded in a user program, it creates a workspace pool for the program's use. You can subsequently use GET W or GET B to access portions of the workspace pool.

When used during EDX system generation, this instruction defines the system storage pool (S$POOL). An alternative to the DEFINE BUFFERPOOL instruction is the DEFINE BRB instruction, which you can use to reduce the size of the supervisor. (See the chapter "Planning Storage Requirements" in the *Design and Installation Guide* for a description of S$POOL and an explanation of how to calculate its size.)

Note that you can code DEFINE WORK or DEFINE POOL instead of DEFINE BUFFERPOOL.

DEFINE BUFFERPOOL Format

[*label*]	**DEFINE BUFFERPOOL** **DEFINE WORK** **DEFINE POOL**	**,SIZE=***size*

DEFINE BUFFERPOOL Operands

Operand	Syntax	Description
label	label	The label to be assigned to the pool. Your subsequent GET W and GET B instructions must refer to this label. If no label is specified, the default label S$POOL is generated. When the instruction defines the system storage pool, you must code S$POOL or take the default.
SIZE=	term	The decimal size, in bytes, of the storage pool to be allocated. The maximum size is 32,767. If you specify 0 or a negative value, only a buffer reference block (BRB) is created; its size field contains 0 or the negative value.

DEFINE BUFFERPOOL Examples

```
1 DEFINE BUFFERPOOL,SIZE=4096

2 POOL   DEFINE WORK,SIZE=2048
```

Example 1, when used during EDX system generation, defines a system storage pool of 4096 bytes. When coded in a user program, it defines a workspace pool with the default label S$POOL.

Example 2 defines a workspace pool with the label POOL.

DEFINE DEVICE—Define Remote Disk Access

DEFINE DEVICE defines the information used to access one or more disks on another Series/1 as if they were on this Series/1. It is used in conjunction with one or more DISK statements that define remote disks. DEFINE DEVICE generates entry point and label $CFPDDB; only one DEFINE DEVICE per assembly is allowed.

The information supplied in DEFINE DEVICE is used to access remote disks only when the program dispatcher is not running in this Series/1. When the program dispatcher is running in this Series/1, access to remote disks is through the paths defined to the program dispatcher. DEFINE DEVICE is required even if remote disks are not accessed until the program dispatcher is running in this Series/1.

DEFINE DEVICE is used only during EDX system generation.

DEFINE DEVICE Format

	DEFINE DEVICE	**,{CUDA=***polladdr* **|** **RINGADR=***ringaddr***}** **,LINE=***lineaddr* **,CELL=***cellid* **,ADDRESS=***diskaddr*

DEFINE DEVICE Operands

Operand	Syntax	Description
CUDA=	term	A 3270 polling address, in the form of a fullword hexadecimal constant, or 0. When access to remote disks is over a multipoint BSC line, specify the polling address of the terminal station that represents the path to the program dispatcher in the Series/1 where the real disks are located. When access to remote disks is over a point-to-point BSC line (Series/1-to-Series/1), specify 0.
RINGADR=	literal	The hexadecimal Local Communications Controller ring address of the Series/1 where the real disks are located.

Operand	Syntax	Description
LINE=	literal	The hexadecimal address of the line in this Series/1 that is used to access the remote disks—either a BSC line or subchannel 0 of a Local Communications Controller attachment.
CELL=	literal	The cell ID of this Series/1.
ADDRESS=	literal	The hexadecimal device address of the remote disk. This can also be a list of addresses in parentheses (for example, ADDRESS=(E1,E2,E3)). The address is the one specified on the DISK statement that defines the remote disk.

DEFINE DEVICE Examples

```
1 DISK DEVICE=4962-3,ADDRESS=E1,VOLNAME=(EDX003)

  DEFINE DEVICE,CUDA=X'C240',LINE=9,CELL=S2,ADDRESS=E1

2 DEFINE DEVICE,RINGADR=4F,LINE=50,CELL=S2,ADDRESS=E1
```

Example 1 shows a DISK statement and the DEFINE DEVICE instruction. The DISK statement defines a remote disk with address E1. The device type must be the same as that of the corresponding real disk. The device address must be a fictitious address; there must be no real disk or other device attached at that address.

The DEFINE DEVICE instruction specifies that when the program dispatcher is not running in this Series/1, the corresponding real disk is accessed by means of a transaction sent from cell S2 to the emulated terminal station with polling address X'C240' on BSC line 9.

Example 2 specifies that when the program dispatcher is not running in this Series/1, the real disk corresponding to the one with address E1 is accessed by means of a transaction sent from cell S2 over the Local Communications Controller attachment at address 50 to the Series/1 with ring address 4F.

DEFINE Q—Define a Queue Control Block

This instruction defines a Communications Facility queue control block.

DEFINE Q Format

label	**DEFINE Q**	

DEFINE Q Operands

Operand	**Syntax**	**Description**
label	label	The label of the queue control block created.

DEFINE Q Return Codes

None.

DEFINE Q Example

QCB1 DEFINE QUE

This instruction defines a queue control block with the label QCB1.

DEFINE VOLUME—Define a Remote Disk Volume

DEFINE VOLUME makes an entry for a remote disk volume in the volume descriptor cross-reference table. The first DEFINE VOLUME statement generates entry point and label $CFPVDE.

DEFINE VOLUME is used only during EDX system generation.

DEFINE VOLUME Format

	DEFINE VOLUME	**,CELL=***cellid* **,VOLNAME=(***local-name,remote-name***)** **[END={YES | NO}]**

DEFINE VOLUME Operands

Operand	**Syntax**	**Description**
CELL=	literal	The cell ID of the Series/1 where the real disk volume is located.
VOLNAME=	literal	The volume name used in this Series/1, followed by the real volume name.
END=	literal	YES if this is the end of the volume descriptor cross-reference table; NO otherwise.

DEFINE VOLUME Examples

```
DEFINE VOLUME,CELL=SJ,VOLNAME=(PSD002,EDX002)
DEFINE VOLUME,CELL=SJ,VOLNAME=(EDX003,EDX003),END=YES
```

These instructions define remote volumes located in cell SJ. The first one defines volume EDX002, which is known as PSD002 in this Series/1. The second one defines the volume known as EDX003 both in this Series/1 and in cell SJ.

The remote volumes and the names by which they are known in this Series/1 must be designated as performance volumes (specified in the VOLNAME operand of a DISK statement).

FREE B—Free a Buffer

This instruction returns a Communications Facility buffer obtained with GET B or GET S to the workspace pool from which it was acquired.

FREE B Format

[*label*]	**FREE B**	,*location* **[,EXIT=***label***]** **[,P1=***name1***]**

FREE B Operands

Pn	Operand	Syntax	Description
P1	*location*	location	The location of a word that contains the address of the buffer. The address was returned to you when you issued the GET B or GET S instruction.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

FREE B Return Codes

-1	Successful.
1	The address specified in the *location* operand was 0.
2	The address specified in the *location* operand is invalid because it is the address of an odd byte.
3	An incorrect forward pointer was found in the workspace pool. Report the problem to your IBM representative.
4	An incorrect backward pointer was found in the workspace pool. Report the problem to your IBM representative.

FREE B Example

```
FREE B,#1
```

This example returns the buffer whose address is in register 1 to the pool from which it was acquired.

FREE S—Free Storage

This instruction returns storage obtained with GET B, GET S, or GET W to the workspace pool from which it was acquired. Note that you can code this instruction as FREE S or FREE W.

FREE S Format

[*label*]	**FREE S** **FREE W**	,*location* **[,TYPE=BUFFER]** **[,EXIT=***label***]** **[,P1=***name1***]**

FREE S Operands

Pn	Operand	Syntax	Description
P1	*location*	location	The location of a word that contains the address of the storage. The address was returned to you when you issued the GET S or GET W instruction.
	TYPE=	literal	Specify TYPE=BUFFER if the storage being returned is a Communications Facility buffer. Note that FREE S with TYPE=BUFFER is the same as FREE B.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

FREE S Return Codes

-1 Successful.

1 The address specified in the *location* operand was 0.

2 The address specified in the *location* operand is invalid because it is the address of an odd byte.

3 An incorrect forward pointer was found in the workspace pool. Report the problem to your IBM representative.

4 An incorrect backward pointer was found in the workspace pool. Report the problem to your IBM representative.

FREE S Example

```
FREE S,#2
```

The storage whose address is in register 2 is freed.

GET A—Locate a System Facility

The GET A instruction allows you to get the addresses of various system facilities.

GET A Format

[*label*]	GET A[6]	,*location* [,{*facility* | **#TCB**}] [,**EXIT**=*label*] [,**P1**=*name1*] [,**P2**=*name2*]

GET A Operands

Pn	Operand	Syntax	Description
P1	*location*	location	The location where the address of the facility is to be returned.
P2	*facility*	integer	A value from Figure 31.
		literal	A literal from Figure 31 .
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

Value	Literal	System Facility
0	#TCB	Task control block
1	#CCB	Terminal control block ($TCBCCB)
2	#BUFFER	Buffer in terminal control block ($CCBBFAD)
3	#LINE	Station block associated with current task control block.
4	LUPOOL	System storage pool (S$POOL)
5	BUFPOOL	System storage pool (S$POOL)
6	SYSQ	System queue control block
7	SYSCOM	EDX system common area ($SYSCOM)
8	CSXTABLE	Language extension command table (S$CMDTBL, in module S$CSXSYS)

Figure 31. System Facilities Available through GET A

6 Any characters *except* I can follow the A. For example, you can enter GET ADDRESS or GET ALBATROSS, but not GET AI or GET AID. This restriction applies because GET AI is another instruction.

GET A Return Codes

-1 Successful.

1 *facility* specified as an integer is invalid; *location* is set to 0.

GET A Examples

```
1            GET A,#2,#LINE

2            GET A,#2,FACILITY*
FACILITY     DATA '0'              INTEGER FOR LITERAL'#TCB'
```

Example 1 requests that the address of the station block associated with the current TCB be returned in register 2. Either the TCB must be within the station block, or the word following the TCB must contain the address of the station block.

Example 2 requests that the address of the current TCB be returned in register 2.

GET AI—Retrieve the AID Byte and Cursor Address from a 3270 Data Stream

GET AI retrieves the attention ID (AID) byte and, optionally, the cursor address from a 3270 data stream that is in a Communications Facility buffer. You can also use this instruction to verify whether the cursor is at a specific row and column position. It is assumed that the 3270 data stream is the result of a 3270 read command. If the data stream is the result of a short read operation, it contains no cursor location.

GET AI Format

[*label*]	**GET AI**	,*text* ,*buffer* [{,*position* | **,ROW=***row***,COLM=***column*}] [**,WIDTH=**{**80** | *width*}] [**,EXIT=***label*] [**,P1=***name1*] [**,P2=***name2*] [**,P3=***name3*]

GET AI Operands

Pn	Operand	Syntax	Description
P1	*text*	location	The address of an EDX text area to receive the AID byte.
P2	*buffer*	location	The address of the Communications Facility buffer that contains the 3270 data stream.
P3	*position*	integer	To verify the cursor position, the screen location to be verified; the valid range is 0-4096. You can also specify the location by using the ROW and COLM operands.
		location	To retrieve the cursor position, the location to receive it. The location must have an address greater than 4096.
	ROW=	term	The cursor row position to be verified. ROW and COLM are used together in place of the *position* operand.
	COLM=	term	The cursor column position to be verified. COLM must be between 1 and the value of WIDTH. ROW and COLM are used together in place of the *position* operand.

Pn	Operand	Syntax	Description
	WIDTH=	term	The screen width. This operand is meaningless unless ROW and COLM are specified.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

GET AI Return Codes

-1 Successful. The AID byte was moved. If *position* was specified to retrieve the cursor location and a cursor location exists in the data stream, it was moved to *position*; if no cursor location exists in the data stream, *position* is set to -1. If cursor verification was specified, the actual cursor location is as specified.

1 Cursor verification was specified, but the buffer contained no cursor location.

3 The cursor verification specification does not describe the actual cursor location.

4 The buffer does not contain a valid 3270 data stream; that is, it doesn't begin with an STX (X'02').

GET AI Example

```
        GET AID,TEXT1,TPBUF,CURPOS
TEXT1   TEXT LENGTH=1
CURPOS  DATA F'0'
```

This example retrieves the AID byte and the cursor location from the data stream in TPBUF.

Assuming that TPBUF contains this data stream:

X'0240407D4CC911C2D3F1F2F3114CC8C1C2C3C403'

the result is:

TEXT1=X'7D' (the AID value)
CURPOS=X'0309' (the cursor location, in binary)

GET B—Get a Buffer

The GET B instruction gets a Communications Facility buffer from a workspace pool in your program.

GET B Format

[*label*]	**GET B**	**,*location*** **,*size*** **[,POOL={*location* | S$POOL}]** **[,WAIT={YES | NO}]** **[,EXIT=*label*]** **[,P1=*name1*]** **[,P2=*name2*]** **[,P3=*name3*]**

GET B Operands

Pn	Operand	Syntax	Description
P1	*location*	location	The location to receive the address of the data area of the buffer.
P2	*size*	integer	The requested size of the buffer, in bytes. This is the size of the data area of the buffer; it doesn't include the buffer header. The size must be in the range 1-32760.
P3	POOL=	location	The address of the pool from which the buffer is to be acquired. If you use the default label S$POOL, it must be defined in your program; the label does not refer to the system storage pool.
	WAIT=	literal	An indicator of whether or not the instruction is to wait until the requested space is available. Don't specify WAIT=YES unless storage is available or another task will release some. Note that if the requested size is greater than the size of the workspace pool and you code WAIT=YES, the instruction will never complete.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

GET B Return Codes

-1 Successful. The buffer address is in *location.*

1 Not enough space is available. *location* is set to 0.

2 The requested buffer size is 0 or negative.

GET B Examples

```
1          GET B,#1,80,POOL=POOL

POOL       DEFINE POOL,SIZE=4000

2          GET BUFFER,#1,#2
           DEFINE POOL,SIZE=1000

3          GET BUFF,#1,256,POOL=POOL@*

POOL@      DATA A(*-*)
```

Example 1 requests an 80-byte buffer from the workspace pool POOL, and waits until the space is available. On return, register 1 contains the address of the buffer.

In example 2, register 2 contains the buffer size you want to be allocated from S$POOL in your program. On return, register 1 contains the address of the buffer.

Example 3 allocates a 256-byte buffer from the pool whose address is in POOL@. This example is related to Figure 1 on page 7.

GET F—Retrieve a Field from a Buffer

The GET F instruction moves data from a Communications Facility buffer to an EDX text area. Depending on the combination of operands chosen, you can use GET F to retrieve a specific amount of text; to retrieve all the text preceding a delimiter; and to get the next sequential field or a specific field from a 3270 data stream. The number of bytes moved is recorded in the text area header. Unused bytes in the text area are set to blanks.

A GET F instruction may complete successfully and return a record of length 0. This means the instruction detected a null field. A null field results from two delimiters in a row with no intervening text, or a delimiter as the last character in the buffer.

GET F Format

[*label*]	**GET F**	,*text* ,*buffer* [,*position*] [,**COMPARE**=*integer*] [,**ROW**=*row*] [,**COLM**={<u>2</u> | *column*}] [,**ATTR**=*location*] [,**TYPE=NUMERIC**] [,**WIDTH**={<u>80</u> | *width*}] [,**EXIT**=*label*] [,**P1**=*name1*] [,**P2**=*name2*] [,**P3**=*name3*]

GET F Operands

Pn	Operand	Syntax	Description
P1	*text*	location	The address of an EDX text area to which the data is to be moved.
P2	*buffer*	location	The address of the buffer from which the data is to be moved.
P3	*position*	integer	For retrieval of a specific field from a 3270 data stream, the screen position of the field data (not of the attribute character); the valid range is 0-4096. Note that you can also specify the position through the ROW and COLM operands.

Pn	Operand	Syntax	Description
		location	For sequential retrieval from a 3270 data stream, the location to receive the screen position of the field. The location must have an address greater than 4096.
P3	COMPARE=	integer	A character that delimits the field, coded in bits 8-15. Note that if you use this operand, you can't use *position* or ROW and COLM operands.
	ROW=	term	For retrieval of a specific field from a 3270 data stream, the row position of the field on the screen. ROW and COLM are used together in place of the *position* operand.
	COLM=	term	For retrieval of a specific field from a 3270 data stream, the column position of the field on the screen. COLM must be between 1 and the value of WIDTH. ROW and COLM are used together in place of the *position* operand.
P4	ATTR=	location	For retrieval from a 3270 data stream, the location to receive the start field order and its associated attribute character. Note that this operand is meaningful only when you retrieve a field from a 3270 write data stream; a data stream that is the result of a read modified operation does not contain start field sequences. If this operand is specified and no start field order exists, *location* is set to 0. If a start field order exists and this operand is not specified, the start field sequence is treated as data and moved to the beginning of *text*.
	TYPE=	literal	NUMERIC specifies that the data is to be right-justified in the text area, with any unused bytes to the left being filled with zeros. This operand is ignored unless *position,* ROW and COLM, or COMPARE is specified.
	WIDTH=	term	The screen width. This operand is meaningless unless ROW= and COLM= are specified.

Pn	Operand	Syntax	Description
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

GET F Return Codes

-1	Successful. The data has been moved or a null field was detected.
1	For get sequential only, the end of the buffer was detected. This is the last data field in the buffer. A subsequent get sequential will result in return code 4.
2	For all gets except get sequential, a text area overrun occurred. The receiving text area was filled and the balance of the field was truncated.
4	For get sequential, get sequential by delimiter, and get 3270 sequential, the instruction requested a read past the end of the data buffer. No data was moved to the text area, and the text count *text*-1 was set to 0. The location specified by *position* was set to -1. For get specific 3270 field, there is no field at the specified position. No data was moved to the text area, and the text count *text*-1 was set to 0.

GET F Examples

1 Get Sequential

```
         GET F,TEXT1,TPBUF
         GET F,TEXT2,TPBUF
         GET F,TEXT3,TPBUF
TEXT1    TEXT LENGTH=2
TEXT2    TEXT LENGTH=3
TEXT3    TEXT LENGTH=2
```

Assuming the following data in TPBUF:

'ABCDEFG'

The result in the text areas is:

Label	Header	Data
TEXT1	X'0202'	'AB'
TEXT2	X'0303'	'CDE'
TEXT3	X'0202'	'FG'

2 Get Sequential by Delimiter

```
         GET FIELD,CPCMD,TPBUF,COMPARE=X'40'
         GET FIELD,CPTEXT,TPBUF,COMPARE=X'40'
CPCMD    TEXT LENGTH=4
CPTEXT   TEXT LENGTH=8
```

Assuming the following data in TPBUF:

'CP TEST'

The result in the text areas (where b represents a blank) is:

Label	Header	Data
CPCMD	X'0402'	'CPbb'
CPTEXT	X'0804'	'TESTbbbb'

3 Get Specific 3270 Field

```
         GET F,FIELD1,TPBUF,ROW=3,COLM=4
         GET F,FIELD2,TPBUF,LOC*
FIELD1   TEXT LENGTH=8
FIELD2   TEXT LENGTH=8
LOC      DATA F'82'
```

Note that screen position 82 is equivalent to row 2, column 3.

Assuming the following data in TPBUF:

X'02C1407D404011C1D2F1F2F311C2E3C1C2C3C403'

The result in the text areas (where b represents a blank) is:

Label	Header	Data
FIELD1	X'0804'	'ABCDbbbb'
FIELD2	X'0803'	'123bbbbb'

4 Get Next Sequential 3270 Field

```
            GET FIELD,TEXT1,TPBUF,POSITON1
            GET FIELD,TEXT2,TPBUF,POSITON2
TEXT1       TEXT LENGTH=8
TEXT2       TEXT LENGTH=8
POSITON1    DATA A(*-*)
POSITON2    DATA A(*-*)
```

Assuming the following data in TPBUF:

X'0240407D404011C2D3F1F2F3114CC8C1C2C3C403'

The result in the text areas and position fields (where b represents a blank) is:

Label	Header	Data
TEXT1	X'0803'	'123bbbbb'
TEXT2	X'0804'	'ABCDbbbb'
POSITON1		F'147'
POSITON2		F'776'

POSITON1 shows that the data appeared from row 2, column 68 (position 147); POSITON2 shows that the data appeared from row 10, column 57 (position 776).

GET Q—Get an Element from a Queue

GET Q gets an element from a queue. Normally (unless you specify PRI= or OPTION=), it gets the first element on the queue. If you specify PRI=, it gets the first element of the specified priority. If you specify OPTION=REMOVE, it removes the specific element whose address you have placed at *element*.

GET Q Format

[*label*]	**GET Q**	,*element* ,*qcb* [,**PRI**=*priority*] [,**OPTION=REMOVE**] [,**EXIT**=*label*] [,**P1**=*name1*] [,**P2**=*name2*]

GET Q Operands

Pn	**Operand**	**Syntax**	**Description**
P1	*element*	location	The location where the address of the element is to be placed. If OPTION=REMOVE is coded, this address must contain the location of the element to be removed.
P2	*qcb*	location	The location of the Communications Facility queue control block, which was created by the DEFINE Q instruction.
	PRI=	integer	The priority of the element to be gotten. The range of valid priorities is 1-127. The default is the first element in the queue, without regard to priority. This operand is invalid if OPTION=REMOVE is coded.
	OPTION=	literal	REMOVE indicates that *element* contains the address of the element to be removed from the queue.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

GET Q Return Codes

-1 Successful. The address of the element is at *element*, or, if OPTION=REMOVE was specified, the element has been removed.

1 The queue is empty. *element* has been set to 0.

2	The QCB address is 0.
3	There is an incorrect forward pointer in the chain of station blocks. Report the problem to your IBM representative.
4	There is an incorrect backward pointer in the chain of station blocks. Report the problem to your IBM representative.

GET Q Example

```
         GET QOUT,#2,QCB1
QCB1     DEFINE QUE
```

In this example, the address of the first element on the queue QCB1 is returned in register 2.

GET S—Get Storage

The GET S instruction gets storage or a Communications Facility buffer from the system storage pool (S$POOL).

GET S Format

[*label*]	**GET S**	,*location* ,*size* **[,WAIT={YES | NO}]** **[,TYPE=BUFFER]** **[,EXIT=***label***]** **[,P1=***name1***]** **[,P2=***name2***]**

GET S Operands

Pn	Operand	Syntax	Description
P1	*location*	location	The location to receive the address of the storage or buffer.
P2	*size*	integer	The requested size of the storage or buffer, in bytes. The size must be in the range 1-32760.
	TYPE=	literal	Specify TYPE=BUFFER if you want the storage acquired to be a Communications Facility buffer. This operand has no other valid values.
	WAIT=	literal	An indicator of whether or not the instruction is to wait until the requested space is available. Don't specify WAIT=YES unless storage is available or another task will release some. Note that if the requested size is greater than the size of the workspace pool and you code WAIT=YES, the instruction will never complete.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

GET S Return Codes

-1	Successful. The address of the storage or the buffer is in *location.*
1	Not enough space is available. *location* is set to 0.
2	The requested storage size is 0 or negative.

GET S Examples

```
1       GET STORAGE,#1,#2,EXIT=NOSTOR,WAIT=NO

2       GET S,LOC,SIZE*,TYPE=BUFFER

SIZE    DATA F'80'
LOC     DATA A(*-*)
```

In example 1, register 2 contains the number of bytes to be allocated. On return, if storage was available, its address is in register 1. If the storage is not available, register 1 contains 0 and execution of the program resumes at location NOSTOR.

In example 2, storage location SIZE contains the size of the buffer to be allocated. On return, storage location LOC contains the address of the buffer. If the storage is not available, the program waits until it is.

GET W—Get Workspace

The GET W instruction gets storage from a workspace pool in your program.

GET W Format

[*label*]	GET W	,*location* ,*size* [,POOL={*location* | S$POOL}] [,WAIT={YES | NO}] [,EXIT=*label*] [,P1=*name1*] [,P2=*name2*] [,P3=*name3*]

GET W Operands

Pn	Operand	Syntax	Description
P1	*location*	location	The location to receive the address of the storage.
P2	*size*	integer	The requested size of the storage, in bytes. The size must be in the range 1-32760.
P3	POOL=	location	The address of the pool from which the storage is to be acquired. If you use the default label S$POOL, it must be defined in your program; the label does not refer to the system storage pool.
	WAIT=	literal	An indicator of whether or not the instruction is to wait until the requested space is available. Don't specify WAIT=YES unless storage is available or another task will release some. Note that if the requested size is greater than the size of the workspace pool and you code WAIT=YES, the instruction will never complete.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

GET W Return Codes

-1	Successful. The storage address is in *location*.
1	Not enough space is available. *location* is set to 0.

2 The requested storage size is 0 or negative.

GET W Examples

```
1 GET W,#1,SIZE*,POOL=POOL

SIZE DATA F'100'

2 GET WORKSPACE,#2,80,WAIT=NO
```

Example 1 gets 100 bytes from the workspace pool labeled POOL. On return, register 1 contains the address of the storage obtained. If the storage is not available, the program waits until it is.

Example 2 gets 80 bytes from a workspace pool named S$POOL in your program. On return, register 2 contains the address of the storage obtained. If storage is not available, the program doesn't wait; in this case, the TCB code word is set to 1 and register 2 is set to 0.

LCC—Define Local Communications Controller Channel

The LCC instruction defines a Local Communications Controller channel to the EDX supervisor. All LCC instructions must appear together in the $EDXDEFS data set. Each Local Communications Controller has three subchannels; all three subchannels must be defined for each Local Communications Controller attachment.

LCC is used only during EDX system generation.

LCC Format

	LCC	**ADDRESS=***chanaddr* **[,END=YES]**

LCC Operands

Operand	Syntax	Description
ADDRESS=	literal	The hexadecimal address of the Local Communications Controller subchannel.
END=	literal	This operand must be included on the last LCC instruction.

LCC Examples

```
LCC ADDRESS=50
LCC ADDRESS=51
LCC ADDRESS=52
LCC ADDRESS=A8
LCC ADDRESS=A9
LCC ADDRESS=AA,END=YES
```

The LCC instructions in this example define two Local Communications Controller attachments, one starting at address X'50' and the other at address X'A8'.

LOCATE NA—Locate a Station Block by Network Address

The LOCATE NA instruction locates a station's control block, given its network address.

LOCATE NA Format

[*label*]	**LOCATE NA**	*,location* *,na* [**,EXIT=***label*] [**,P1=***name1*] [**,P2=***name2*]

LOCATE NA Operands

Pn	Operand	Syntax	Description
P1	*location*	location	The location to receive the address of the station block.
P2	*na*	location	The location that contains the network address of the station.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

LOCATE NA Return Codes

-1	The station was found; the address of the station block is at *location.*
1	The station was not found; *location* is set to 0.

LOCATE NA Example

```
LOCATE NA,#1,(Q$DLV,#2)
```

In this example, the address of a station's control block is in register 2 and the network address of the station to which it is linked is at (Q$DLV,#2). If the linked-to station is started, the address of its station block is returned in register 1. If the station isn't started, 0 is returned in register 1.

LOCATE ST—Create, Delete, Purge, or Locate a Station Block

The LOCATE ST instruction locates a station's control block, given its station name. It also allows you to delete a station's control block, or create a station control block. Note that you can code LOCATE LU or LOCATE QN instead of LOCATE ST.

LOCATE ST Format

[*label*]	**LOCATE ST** **LOCATE LU** **LOCATE QN**	,*location* [,*staname*] **[,TYPE={2 | *type*}]** **[,WORK={0 | *size*}]** **[,OPTION={CREATE | REMOVE** **| DELETE | PURGE | PROGSTOP}]** **[,TCB=NO]** **[,EXIT=***label***]** **[,P1=***name1***]** **[,P2=***name2***]** **[,P3=***name3***]**

LOCATE ST Operands

Pn	Operand	Syntax	Description
P1	*location*	location	The location to receive the address of the station block.
P2	*staname*	location	The location of the 8-character literal name of the station. The default is the name of this program.
		'string'	The station name, enclosed in quotes. A name longer than 8 characters is truncated to 8 characters.
P3	TYPE=	integer	2 for a user station; 12 for a message station. The values for all station types are given in the *Debugging Guide*.
		literal	USER or PROGRAM for a user station; MESSAGE for a message station. The literals for all station types are given in the *Debugging Guide*. TYPE is valid only for OPTION=CREATE.

Pn	Operand	Syntax	Description
	WORK=	term	The amount of workspace to be allocated when the station block is created. This value is specified in bytes, from 0 to 255. It is valid only for OPTION=CREATE. Workspace is allocated immediately following the station block. The address of the workspace is placed in field Q$WORK in the station block.
	OPTION=	literal	One of the following values: CREATE—Create the station block if one named *staname* does not already exist. REMOVE or DELETE—Delete the station block if there are no storage-queued messages on the station's queue. The station block address is *not* returned in *location.* PURGE—Remove storage-queued messages from the station's queue and delete its station block. The station block address is *not* returned in *location.* PROGSTOP—Remove storage-queued messages from the station's queue, delete its station block, and perform a PROGSTOP. The station block address is *not* returned in *location.*
	TCB=	literal	This operand is valid only with OPTION=CREATE. NO indicates that the station will be associated with a task control block that is not contained within the station block.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

LOCATE ST Return Codes

-2	The station was not found and OPTION=CREATE was specified; the station block was created and its address is at *location.*
-1	The station was found; unless the station block was deleted, its address is at *location.*
1	The station was not found and OPTION=CREATE was not specified. No action was taken; 0 is returned at *location.*[7]
2	The station was not found and OPTION=CREATE was specified, but no storage was available in the system storage pool. 0 is returned at *location.*[7]
3	There is an incorrect forward pointer in the chain of station blocks. Report the problem to your IBM representative.
4	There is an incorrect backward pointer in the chain of station blocks. Report the problem to your IBM representative.
5	REMOVE or DELETE was specified, but the station wasn't deleted because it has storage-queued messages pending. Either remove all the storage-queued messages or use OPTION=PURGE.

LOCATE ST Examples

```
1 LOCATE ST,#1,OPTION=CREATE

2 LOCATE LU,#1,'PROGA'
```

In example 1, if a station block with the name of the program issuing the instruction does not exist, one is created; its type is USER and its network address is the station block address. In either case, the address of the station block is returned in register 1.

In example 2, the address of the station block for station PROGA is returned in register 1. If the station block does not exist, 0 is returned in register 1.

7 Note that, if the OPTION operand didn't specify that the station be deleted, you can check for successful completion by testing *location* for a nonzero value.

MOV—Move Data

The MOV instruction moves data from one storage location to another.

MOV Format

[*label*]	**MOV**	,*to* ,*from* [,{*length* | **DWORD** | **BYTE** | **WORD**}] [,**P1**=*name1*] [,**P2**=*name2*] [,**P3**=*name3*]

MOV Operands

Pn	Operand	Syntax	Description
P1	*to*	location	The location to which the data is to be moved.
P2	*from*	location	The location from which the data is to be moved. The data cannot be immediate data.
P3	*length*	integer	The number of bytes to be moved.
		literal	BYTE for one byte, WORD for one word, or DWORD for one doubleword.

MOV Return Codes

None.

MOV Examples

```
1 MOV STATUS,(0,#T)

2 MOV TEXT,(0,#2),#1*
```

Example 1 moves the TCB code word to the location STATUS. It is not necessary to know the address of the TCB, because #T refers to it.

Example 2 moves data from the address pointed to by register 2 to the text area defined by TEXT. The number of bytes is in register 1.

PUT AID—Put an AID Byte into a 3270 Data Stream

The PUT AID instruction moves a 3270 read header that includes an attention ID (AID) byte and, optionally, a cursor address, into a Communications Facility buffer. The data moved into the buffer is X'024040*xxyyyy*', where *xx* is the AID byte you supply and *yyyy* is the optional cursor address.

The buffer header fields B$COUNT and B$DATA@ are reset to indicate that the buffer is empty. Then the data is moved to the beginning of the buffer, and B$COUNT and B$DATA@ are updated.

PUT AID Format

[*label*]	**PUT AID**	,*buffer* ,*text* [{,*position* | **,ROW**=*row*,**COLM**=*column*}] [**,WIDTH**={**80** | *width*}] [**,EXIT**=*label*] [**,P1**=*name1*] [**,P2**=*name2*] [**,P3**=*name3*]

PUT AID Operands

Pn	Operand	Syntax	Description
P1	*buffer*	location	The address of the buffer to which the read header is to be moved.
P2	*text*	location	The address of an EDX text area that contains the AID byte. The text area must be formatted as shown in the first example.
		'string'	One EBCDIC character, enclosed in quotes.
P3	*position*	integer	The screen location of the cursor. If the location is not specified, either here or through the ROW and COLM operands, no cursor address is moved to the buffer.
	ROW=	term	The row position of the cursor. ROW and COLM are used together in place of the *position* operand.
	COLM=	term	The column position of the cursor. COLM must be between 1 and the value of WIDTH. ROW and COLM are used together in place of the *position* operand.

Pn	Operand	Syntax	Description
	WIDTH=	term	The screen width. This operand is meaningless unless ROW and COLM are specified.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

PUT AID Return Codes

-1	Successful. The read header has been moved into the buffer.
2	The *text* address is 0.
3	A buffer overrun occurred. Not enough space remains in the buffer to accommodate the read header; no data was moved.

PUT AID Examples

1
```
        PUT AID,TPBUF,AID
        DC X'0201'                        TEXT COUNT FIELD
AID     DC X'6D02'                        TEXT
```

2
```
        PUT AID,TPBUF,'1',ROW=2,COLM=10
```

Example **1** moves X'0240406D' into the buffer TPBUF. This is the header that would result from a 3270 read operation ended by the CLEAR key.

Example **2** moves X'024040F1C1D9' into the buffer TPBUF. This is the header that would result from a 3270 read operation ended by the PF1 key when the cursor was at row 2, column 10. X'F1' is the AID byte, and X'C1D9' is the 3270 buffer address for row 2, column 10.

PUT CO—Put a WRITE Command into a 3270 Data Stream

The PUT CO instruction moves a 3270 write command and write control character into a Communications Facility buffer. The default data that is moved into the buffer is X'0227F1C3', where X'F1' is a WRITE command and X'C3' is a write control character that resets modified data tags and unlocks the keyboard. Options of this instruction allow you to make the command an ERASE/WRITE or an ERASE/WRITE ALTERNATE, and to change the write control character.

The buffer header fields B$COUNT and B$DATA@ are reset to indicate that the buffer is empty. Then the data is moved to the beginning of the buffer, and B$COUNT and B$DATA@ are updated.

PUT CO Format

[*label*]	**PUT CO**	,*buffer* [**,OPTION=**(*option1,option2,...*)] [**,EXIT=***label*] [**,P1=***name1*]

PUT CO Operands

Pn	**Operand**	**Syntax**	**Description**
P1	*buffer*	location	The address of the buffer to which the command is to be moved.
	OPTION=	literal	TAB puts a program tab order (X'05') into the buffer following the write control character. ERASE makes the command that is moved into the buffer an ERASE/WRITE (X'F5') rather than a WRITE. ALTERNATE makes the command that is moved into the buffer an ERASE/WRITE ALTERNATE (X'7E') rather than a WRITE. Seven additional options allow you to modify the write control character that is put into the buffer: MDT prevents the resetting of modified data tags. LOCK prevents unlocking of the keyboard.

Pn	Operand	Syntax	Description
			TONE causes the alarm to be sounded.
			P40 sets a 40-character print line and prints the contents of the screen.
			P64 sets a 64-character print line and prints the contents of the screen.
			P80 sets an 80-character print line and prints the contents of the screen.
			PRINT sets unformatted print mode and prints the contents of the screen.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

PUT CO Return Codes

-1 Successful. The command has been moved into the buffer.

3 A buffer overrun occurred. Not enough space remains in the buffer to accommodate the command. No data was moved.

PUT CO Example

```
PUT COMMAND,TPBUF,OPTION=(ERASE,PRINT)
```

This instruction moves X'0227F54B' into the buffer TPBUF. X'F5' is an ERASE/WRITE command, and X'4B' is a write control character that initiates a printout operation, resets modified data tags, and unlocks the keyboard.

PUT CURS—Put a Cursor into a 3270 Data Stream

The PUT CURS instruction moves a 3270 insert cursor order (X'13') and, optionally, the position at which the cursor is to be displayed, into a Communications Facility buffer.

The data is moved to the next available location in the buffer, and buffer header fields B$COUNT and B$DATA@ are updated.

PUT CURS Format

[*label*]	**PUT CURS**	***,buffer*** **[{,,*position*[8] |** **,ROW=*row*,COLM=*column*}]** **[,WIDTH={80 | *width*}]** **[,OPTION=TAB]** **[,EXIT=*label*]** **[,P1=*name1*]** **[,P3=*name3*]**

PUT CURS Operands

Pn	Operand	Syntax	Description
P1	*buffer*	location	The address of the buffer to which the insert cursor order is to be moved.
P3	*position*	integer	The screen location of the cursor. If the location is not specified, either here or through the ROW and COLM operands, no cursor address is moved to the buffer.
	ROW=	term	The row position of the cursor. ROW and COLM are used together in place of the *position* operand.
	COLM=	term	The column position of the cursor. COLM must be between 1 and the value of WIDTH. ROW and COLM are used together in place of the *position* operand.
	WIDTH=	term	The screen width. This operand is meaningless unless ROW and COLM are specified.

8 Even though this instruction has only two positional operands, the Communications Facility treats *position* as the third positional operand. Therefore you must code two commas before *position*. If you enter a value for the second positional operand, it is ignored.

Pn	Operand	Syntax	Description
	OPTION=	literal	TAB puts a program tab order (X'05') into the buffer following the insert cursor order.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

PUT CURS Return Codes

-1	Successful. The insert cursor order has been moved into the buffer.
3	A buffer overrun occurred. Not enough space remains in the buffer to accommodate the insert cursor order; no data was moved.

PUT CURS Examples

```
1 PUT CURSOR,TPBUF,ROW=6,COLM=8,OPTION=TAB

2 PUT CURS,TPBUF
```

Example 1 moves X'11C6D71305' into the buffer TPBUF. X'11' is a set buffer address order; X'C6D7' is the 3270 buffer address for row 6, column 8; X'13' is an insert cursor order; and X'05' is a program tab order.

Example 2 moves an insert cursor order (X'13') into the buffer TPBUF. The cursor position depends on previous set buffer address orders and data in the 3270 data stream.

PUT DLEETB—Put a DLE and an ETB into a 3270 Data Stream

The PUT DLEETB instruction moves BSC control characters DLE and ETB (X'1026') into a Communications Facility buffer.

The data is moved to the next available location in the buffer, and buffer header fields B$COUNT and B$DATA@ are updated. This instruction allows the last byte of the buffer to be used.

PUT DLEETB Format

[*label*]	**PUT DLEETB**	,*buffer* [,**EXIT**=*label*] [,**P1**=*name1*]

PUT DLEETB Operands

Pn	**Operand**	**Syntax**	**Description**
P1	*buffer*	location	The address of the buffer to which the DLE and ETB are to be moved.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

PUT DLEETB Return Codes

-1	Successful. The DLE and ETB have been moved into the buffer.
3	A buffer overrun occurred. Not enough space remains in the buffer to accommodate the DLE and ETB. No data was moved.

PUT DLEETB Example

```
PUT DLEETB,TPBUF
```

This instruction puts a DLE and an ETB into the buffer TPBUF.

PUT DLEETX—Put a DLE and an ETX into a 3270 Data Stream

The PUT DLEETX instruction moves BSC control characters DLE and ETX (X'1003') into a Communications Facility buffer.

The data is moved to the next available location in the buffer and buffer header fields B$COUNT and B$DATA@ are updated. This instruction allows the last byte of the buffer to be used.

PUT DLEETX Format

[*label*]	**PUT DLEETX**	**,***buffer* **[,EXIT=***label***]** **[,P1=***name1***]**

PUT DLEETX Operands

Pn	Operand	Syntax	Description
P1	*buffer*	location	The address of the buffer to which the DLE and ETX are to be moved.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

PUT DLEETX Return Codes

-1	Successful. The DLE and ETX have been moved into the buffer.
3	A buffer overrun occurred. Not enough space remains in the buffer to accommodate the DLE and ETX; no data was moved.

PUT DLEETX Example

```
PUT DLEETX,TPBUF
```

This instruction puts a DLE and an ETX into the buffer TPBUF.

PUT DLESTX—Put a DLE and an STX into a 3270 Data Stream

The PUT DLESTX instruction moves BSC control characters DLE and STX (X'1002') into a Communications Facility buffer.

The buffer header fields B$COUNT and B$DATA@ are reset to indicate that the buffer is empty. Then the data is moved to the beginning of the buffer, and B$COUNT and B$DATA@ are updated.

PUT DLESTX Format

[*label*]	**PUT DLESTX**	,*buffer* **[,EXIT=***label***]** **[,P1=***name1***]**

PUT DLESTX Operands

Pn	Operand	Syntax	Description
P1	*buffer*	location	The address of the buffer to which the DLE and STX are to be moved.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

PUT DLESTX Return Codes

-1	Successful. The DLE and STX have been moved into the buffer.
3	A buffer overrun occurred. Not enough space remains in the buffer to accommodate the DLE and STX; no data was moved.

PUT DLESTX Example

```
PUT DLESTX,TPBUF
```

This instruction puts a DLE and an STX into the buffer TPBUF.

PUT ERA—Put an Erase Order into a 3270 Data Stream

The PUT ERA instruction moves a 3270 erase unprotected to address (EUA) order into a Communications Facility buffer. The data moved into the buffer is X'12*xxxx*', where *xxxx* is the 3270 buffer address at which the erase operation is to stop.

The data is moved to the next available location in the buffer, and buffer header fields B$COUNT and B$DATA are updated.

PUT ERA Format

[*label*]	**PUT ERA**	***,buffer*** **{„*position*[9] |** **,ROW=*row*,COLM=*column*}** **[,WIDTH={<u>80</u> | *width*}]** **[,OPTION=TAB]** **[,EXIT=*label*]** **[,P1=*name1*]** **[,P3=*name3*]**

PUT ERA Operands

Pn	Operand	Syntax	Description
P1	*buffer*	location	The address of the buffer to which the erase order is to be moved.
P3	*position*	integer	The screen location at which the erase operation is to stop. Specify the location either here or through the ROW and COLM operands.
	ROW=	term	The row position at which the erase operation is to stop. ROW and COLM are used together in place of the *position* operand.
	COLM=	term	The column position at which the erase operation is to stop. COLM must be between 1 and the value of WIDTH. ROW and COLM are used together in place of the *position* operand.

[9] Even though this instruction has only two positional operands, the Communications Facility treats *position* as the third positional operand. Therefore you must code two commas before *position*. If you enter a value for the second positional operand, it is ignored.

Pn	Operand	Syntax	Description
	WIDTH=	term	The screen width. This operand is meaningless unless ROW and COLM are specified.
	OPTION=	literal	TAB puts a program tab order (X'05') into the buffer following the erase order.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

PUT ERA Return Codes

-1	Successful. The erase order has been moved into the buffer.
3	A buffer overrun occurred. Not enough space remains in the buffer to accommodate the erase order. No data was moved.

PUT ERA Example

```
PUT ERASE,TPBUF,ROW=7,COLM=1
```

This instruction moves X'12C760' into the buffer TPBUF. X'12' is an erase unprotected to address order, and X'C760' is the 3270 buffer address for row 7, column 1.

PUT ETB—Put an ETB into a 3270 Data Stream

The PUT ETB instruction moves BSC control character ETB (X'26') into a Communications Facility buffer.

The data is moved to the next available location in the buffer, and buffer header fields B$COUNT and B$DATA@ are updated. This instruction allows the last byte of the buffer to be used.

PUT ETB Format

[*label*]	**PUT ETB**	,*buffer* [,**EXIT**=*label*] [,**P1**=*name1*]

PUT ETB Operands

Pn	**Operand**	**Syntax**	**Description**
P1	*buffer*	location	The address of the buffer to which the ETB is to be moved.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

PUT ETB Return Codes

-1	Successful. The ETB has been moved into the buffer.
3	A buffer overrun occurred. Not enough space remains in the buffer to accommodate the ETB. No data was moved.

PUT ETB Example

```
PUT ETB,TPBUF
```

This instruction puts an ETB into the buffer TPBUF.

PUT ETX—Put an ETX into a 3270 Data Stream

The PUT ETX instruction moves BSC control character ETX (X'03') into a Communications Facility buffer.

The data is moved to the next available location in the buffer, and buffer header fields B$COUNT and B$DATA@ are updated. This instruction allows the last byte of the buffer to be used.

PUT ETX Format

[*label*]	**PUT ETX**	,*buffer* [**,EXIT=***label*] [**,P1=***name1*]

PUT ETX Operands

Pn	Operand	Syntax	Description
P1	*buffer*	location	The address of the buffer to which the ETX is to be moved.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

PUT ETX Return Codes

-1	Successful. The ETX has been moved into the buffer.
3	A buffer overrun occurred. Not enough space remains in the buffer to accommodate the ETX. No data was moved.

PUT ETX Example

```
PUT ETX,TPBUF
```

This instruction puts an ETX into the buffer TPBUF.

PUT F—Put Data into a Communications Facility Buffer

The PUT F instruction moves data from an EDX text area to a Communications Facility buffer. Depending on the combination of operands chosen, you can use PUT F to concatenate several text fields into one buffer, or to construct a 3270 data stream.

PUT F Format

[*label*]	**PUT F**	,*buffer* ,*text* [,*position*] [,**ATTR**=*integer*] [,**ROW**=*row*] [,**COLM**={<u>**2**</u> | *column*}] [,**WIDTH**={<u>**80**</u> | *width*}] [,**OPTION**={<u>**INITIAL**</u> | **FINAL** | **TAB**}] [,**EXIT**=*label*] [,**P1**=*name1*] [,**P2**=*name2*] [,**P3**=*name3*] [,**P4**=*name4*]

PUT F Operands

Pn	Operand	Syntax	Description
P1	*buffer*	location	The address of the buffer to which the data is to be moved.
P2	*text*	location	The address of an EDX text area from which the data is to be moved.
		'string'	From 1 to 253 EBCDIC characters, enclosed in quotes.
P3	*position*	integer	The location (0-1919) where the field is to be displayed on the receiving device. If the position is not specified, either here or through the ROW and COLM operands, no new field is created; the data is concatenated to any data already in the buffer.

Pn	Operand	Syntax	Description
			If the position is specified, a 3270 set buffer address order is moved to the buffer preceding the field. The order is X'11*xxxx*', where *xxxx* is the 3270 buffer address for the field location. If the ATTR operand is not coded, *xxxx* is the position you specify. If the ATTR operand is coded, *xxxx* is *position*-1, unless *position* is 0; in that case, *xxxx* is location 1919.
P4	**ATTR=**	integer	The attribute character of the field, coded in bits 8-15. If ATTR is coded, either *position* or ROW= and COLM= must be coded. A 3270 start field order is moved to the buffer following the set buffer address order. The order is X'1D*xx*', where *xx* is the attribute character you specify.
	ROW=	term	The row position at which the data is to be displayed on the receiving device. ROW and COLM are used together in place of the *position* operand.
	COLM=	term	The column position at which the data is to be displayed on the receiving device. COLM must be between 1 and the value of WIDTH. ROW and COLM are used together in place of the *position* operand.
	WIDTH=	term	The screen width. This operand is meaningless unless ROW= and COLM= are specified.
	OPTION=	literal	INITIAL specifies that this is the first data in the buffer. This operand resets buffer header fields B$COUNT and B$DATA@ to indicate that the buffer is empty before the data is moved into it. Be sure to code OPTION=INITIAL when you're reusing a buffer. If you don't code OPTION=INITIAL, the data is moved to the next available location in the buffer, and buffer header fields B$COUNT and B$DATA@ are updated.

Pn	Operand	Syntax	Description
			FINAL specifies that this is the last data in the buffer. Specifying FINAL allows you to use the last byte of the buffer, which you can't use otherwise. FINAL is normally used to add an ETB or ETX to an otherwise full buffer.
			TAB puts a program tab order (X'05') into the buffer following the data.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

PUT F Return Codes

-1 Successful. The data has been moved into the buffer.

2 The *text* address is 0.

3 A buffer overrun occurred. Not enough space remains in the buffer to accommodate the data. No data was moved.

PUT F Examples

```
1       PUT F,TPBUF,ERASE,OPTION=INITIAL

2       PUT F,TPBUF,'EXAMPLE',ROW=2,COLM=3,ATTR=X'F0'

3       PUT F,TPBUF,ETX,OPTION=FINAL

         DATA F'1'                TEXT COUNT FIELD
ETX      DATA X'03'               TEXT
         DATA F'4'                TEXT COUNT FIELD
ERASE    DATA X'0227F5C3'         TEXT
```

Figure 32 on page 248 shows the buffer content at each of four stages: empty, and after each of the three PUT F instructions.

Label	Address	Empty	After 1	After 2	After 3
B$COUNT	00F8	00	04	10	11
B$ADDR	00FA	0100	0100	0100	0100
B$DATA@	00FC	0100	0104	0110	0111
	00FE				
TPBUF	0100	40	02	02	02
	0101	40	27	27	27
	0102	40	F5	F5	F5
	0103	40	C3	C3	C3
	0104	40	40	11	11
	0105	40	40	C1	C1
	0106	40	40	D1	D1
	0107	40	40	1D	1D
	0108	40	40	F0	F0
	0109	40	40	C5	C5
	010A	40	40	E7	E7
	010B	40	40	C1	C1
	010C	40	40	D4	D4
	010D	40	40	D7	D7
	010E	40	40	D3	D3
	010F	40	40	C5	C5
	0110	40	40	40	03

Figure 32. PUT F Example

PUT NUL—Set Buffer Address

The PUT NUL instruction moves a 3270 set buffer address (SBA) order into a Communications Facility buffer. The data moved into the buffer is X'11*xxxx*', where *xxxx* is a 3270 buffer address. If you specify neither a buffer address nor OPTION=TAB, no data is moved.

The data is moved into the next available location in the buffer, and buffer header fields B$COUNT and B$DATA@ are updated.

PUT NUL Format

[*label*]	**PUT NUL**	***,buffer*** **[{,,*position*[10] |** **,ROW=*row*,COLM=*column*}]** **[,WIDTH={<u>80</u> | *width*}]** **[,OPTION=TAB]** **[,EXIT=*label*]** **[,P1=*name1*]** **[,P3=*name3*]**

PUT NUL Operands

Pn	Operand	Syntax	Description
P1	*buffer*	location	The address of the buffer in which the address is to be set.
P3	*position*	integer	The screen location at which the 3270 buffer address is to be set. If the location is not specified, either here or through the ROW and COLM operands, no set buffer address order is moved to the buffer.
	ROW=	term	The row position at which the 3270 buffer address is to be set. ROW and COLM are used together in place of the *position* operand.
	COLM=	term	The column position at which the 3270 buffer address is to be set. COLM must be between 1 and the value of WIDTH. ROW and COLM are used together in place of the *position* operand.

[10] Even though this instruction has only two positional operands, the Communications Facility treats *position* as the third positional operand. Therefore you must code two commas before *position*. If you enter a value for the second positional operand, it is ignored.

Pn	Operand	Syntax	Description
	WIDTH=	term	The screen width. This operand is meaningless unless ROW and COLM are specified.
	OPTION=	literal	TAB puts a program tab order (X‘05’) into the buffer following the set buffer address order.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

PUT NUL Return Codes

-1	Successful. The data has been moved into the buffer.
3	A buffer overrun occurred. Not enough space remains in the buffer to accommodate the data. No data was moved.

PUT NUL Example

```
PUT NULL,TPBUF,ROW=7,COLM=1
```

This instruction moves X‘11C760’ into the buffer TPBUF. X‘11’ is a set buffer address order, and X‘C760’ is the 3270 buffer address for row 7, column 1.

PUT Q—Put an Element in a Queue

PUT Q puts an element onto a queue.

PUT Q Format

[*label*]	**PUT Q**	,*qcb* ,*element* [,{*priority* | <u>127</u> | YES}] [,P1=*name1*] [,P2=*name2*] [,P3=*name3*]

PUT Q Operands

Pn	Operand	Syntax	Description
P1	*qcb*	location	The location of the Communications Facility queue control block (created through DEFINE Q).
P2	*element*	location	The location that contains the address of the element to be added to the queue.
P3	*priority*	integer	The priority of the element, where 1 is the highest priority.
		literal	YES specifies that the user has provided the priority in the queue element priority byte.
			If the *priority* operand is not coded, the element is placed at the end of the queue.

PUT Q Return Codes

None.

PUT Q Examples

```
1               PUT QIN,QCB1,ADDR1

2               PUT QIN,QCB2,ADDR2,YES

3               PUT QIN,QCB2,ADDR3,4

QCB1            DEFINE QUE
QCB2            DEFINE Q
ADDR1           DATA A(ELEMENT1)
ADDR2           DATA A(ELEMENT2)
ADDR3           DATA A(ELEMENT3)
ELEMENT1        DATA 2F'0',...
ELEMENT2        DATA 2F'0',X'13',X'00'...
ELEMENT3        DATA 3F'0',...
```

Example 1 places ELEMENT1 into QCB1 in first-in-first-out priority.

Example 2 places ELEMENT2 into QCB2 according to its priority. The priority is contained in the fifth byte of the element; it is 19 (decimal). The element is put into the queue after other elements of priority 19 and before any elements of priority 20.

Example 3 inserts ELEMENT3 into QCB2 with priority 4. The element is inserted ahead of ELEMENT2. The fifth byte will be set to X'04'.

PUT REP—Repeat a Character in a 3270 Data Stream

The PUT REP instruction moves a 3270 repeat to address (RA) order into a Communications Facility buffer. The data moved into the buffer is X'3C*xxxxyy*', where *xxxx* is the 3270 buffer address at which the repeat operation is to stop, and *yy* is the repeated character.

The data is moved to the next available location in the buffer, and buffer header fields B$COUNT and B$DATA@ are updated.

PUT REP Format

[*label*]	**PUT REP**	,*buffer* ,*character* {,*position* | ,**ROW**=*row*,**COLM**=*column*} [,**WIDTH**={**80** | *width*}] [,**OPTION=TAB**] [,**EXIT**=*label*] [,**P1**=*name1*] [,**P2**=*name2*] [,**P3**=*name3*]

PUT REP Operands

Pn	**Operand**	**Syntax**	**Description**
P1	*buffer*	location	The address of the buffer to which the repeat order is to be moved.
P2	*character*	location	The address of an EDX text area that contains the repeated character. The text area must be formatted as shown in example **1**.
		'string'	One EBCDIC character, enclosed in quotes.
P3	*position*	integer	The screen location at which the repeat operation is to stop. Specify the location here or through the ROW and COLM operands.
	ROW=	term	The row position at which the repeat operation is to stop. ROW and COLM are used together in place of the *position* operand.

Pn	Operand	Syntax	Description
	COLM=	term	The column position at which the repeat operation is to stop. COLM must be between 1 and the value of WIDTH. ROW and COLM are used together in place of the *position* operand.
	WIDTH=	term	The screen width. This operand is meaningless unless ROW and COLM are specified.
	OPTION=	literal	TAB puts a program tab order (X'05') into the buffer following the repeat to address order.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

PUT REP Return Codes

-1	Successful. The repeat order has been moved into the buffer.
2	The *character* address is 0.
3	A buffer overrun occurred. Not enough space remains in the buffer to accommodate the repeat order. No data was moved.

PUT REP Examples

1
```
        PUT REP,TPBUF,CHAR,ROW=2,COLM=1
         DC X'0201'                TEXT COUNT FIELD
CHAR     DC X'403C'                TEXT
```

2
```
        PUT REPEAT,TPBUF,'*',ROW=2,COLM=1
```

Example **1** moves X'3CC15040' into the buffer TPBUF. X'3C' is a repeat to address order; X'C150' is the 3270 buffer address for row 2, column 1; and X'40' is the repeated character, a blank.

Example **2** has the same effect as example **1**, except that the repeated character is an asterisk.

PUT STX—Put an STX into a 3270 Data Stream

The PUT STX instruction moves BSC control character STX (X'02') into a Communications Facility buffer.

The buffer header fields B$COUNT and B$DATA@ are reset to indicate that the buffer is empty. Then the data is moved to the beginning of the buffer, and B$COUNT and B$DATA@ are updated.

PUT STX Format

[*label*]	**PUT STX**	,*buffer* [**,EXIT=***label*] [**,P1=***name1*]

PUT STX Operands

Pn	**Operand**	**Syntax**	**Description**
P1	*buffer*	location	The address of the buffer to which the STX is to be moved.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

PUT STX Return Codes

-1	Successful. The STX has been moved into the buffer.
3	A buffer overrun occurred. Not enough space remains in the buffer to accommodate the STX. No data was moved.

PUT STX Example

```
PUT STX,TPBUF
```

This instruction puts an STX into the buffer TPBUF.

PUT TCB—Create a Task Control Block

The PUT TCB instruction creates a task control block.

PUT TCB Format

[*label*]	**PUT TCB**	,*tcb* [,*station*] [,**LEV**={<u>2</u> | *level*},**PRI**={<u>150</u> | *priority*}] [,**EXIT**=*label*]

PUT TCB Operands

Pn	Operand	Syntax	Description
P1	*tcb*	location	The address of the area to contain the TCB. The area must be 130 bytes long if *station* is specified and 128 bytes long if *station* is not specified.
P2	*station*	location	The address of the station block with which the TCB is to be associated. If this operand is omitted, the TCB is not associated with a station.
P3	**LEV=**	integer	The hardware interrupt level at which the task is to execute; 1, 2, or 3. This operand is ignored if *station* is specified.
P4	**PRI=**	integer	The priority at which the task is to execute; 1 to 255. This operand is ignored if *station* is specified.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

PUT TCB Return Codes

-1	Successful.
1	The specified station block address or TCB area address is 0.
2	The specified station block is not large enough to be associated with a TCB.
3	The specified level or priority is invalid.

PUT TCB Examples

```
PUT TCB,#1*,#2*
```

This instruction creates a TCB in the area whose address is in register 1 that is associated with the station block whose address is in register 2.

```
PUT TCB,TCB2*,LEV=2,PRI=255
```

This instruction creates a TCB in the area whose address is in TCB2 that is not associated with a station block.

RECEIVE M—Receive a Message into a Buffer

The RECEIVE M instruction receives a message from a station's queue into a Communications Facility buffer. On successful completion, the message length is in buffer header field B$COUNT.

RECEIVE M Format

[*label*]	**RECEIVE M**	*,buffer* [,*staname*] **[,WAIT={<u>YES</u> | NO}]** **[ACK={<u>YES</u> | NO}]** **[,{HEADER=***location* **|** **ORIGIN=***location***}]** **[,OPTION={COPY | KEEP}]** **[,EXIT=***label***]** **[,P1=***name1***]** **[,P2=***name2***]** **[,P3=***name3***]**

RECEIVE M Operands

Pn	**Operand**	**Syntax**	**Description**
P1	*buffer*	location	The address of the buffer into which the message is to be placed.
P2	*staname*	location	The location of the 8-character name of the station from whose queue the message is to be received. The default is the name of the program that issues the instruction.
		'string'	The station name, enclosed in quotes.
	WAIT=	literal	An indicator of whether or not the instruction is to wait until there is a message on the queue. If the station is deleted while a wait is in effect, the instruction returns a return code of 2.
	ACK=	literal	An indicator of whether or not receipt of the message is to be acknowledged to the sender of the message. ACK=YES has no effect if the sender is in a different node.

Pn	Operand	Syntax	Description
P3	HEADER=	location	The address at which the message header is to be placed. The area at *location* must be at least 24 bytes long to accommodate the header.
P3	ORIGIN=	location	The address at which the name of the station that originated the message is to be placed. The area at *location* must be at least 10 bytes long. Bytes 1 to 8 contain the station name, and bytes 9 and 10 contain the station type and subtype. If the origin of the message is not known, the area at *location* is filled in with binary zeros.
	OPTION=	literal	COPY indicates that a copy of the message is to be delivered normally, but the message is to be left on the queue. KEEP has the same effect as COPY, and also indicates that if the receiving program's buffer is too small to hold a disk-queued message, a copy of the message is to be kept in the system message buffer pool.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

RECEIVE M Return Codes

-19 The receive completed successfully, and the disk-queue data set's capacity warning level, specified when the data set was defined, has been exceeded.

-18 The receive completed successfully, and an overlay occurred in the disk-queue data set.

-1 Successful. A message has been moved to the buffer at *buffer*.

1 There is no message on the queue and WAIT=NO was specified.

2 The station specified by *staname* does not exist.

3 The buffer at *buffer* isn't big enough to contain the message; the message has been truncated. If OPTION=KEEP was specified, the buffer header count field contains the message length; otherwise it contains the number of bytes moved to the buffer.

4 Messages for *staname* are being held as a result of a CP F command that set output hold. No message was received.

5 The buffer address specified was 0.

6 A status message (one sent by a SEND Status instruction) was received.

7 The message to be received was disk-queued and a disk I/O error occurred. No message was received.

8 The message to be received was disk-queued and the station specified by *staname* is not active. No message was received.

9 The previous receive from the queue specified by *staname* specified OPTION=COPY or KEEP, and no RECEIVE P instruction has been issued. No message was received.

RECEIVE M Examples

```
1         RECEIVE MESSAGE,TPBUF,ORIGIN=ORIGIN

TPBUF     DEFINE BUFFER,SIZE=256
ORIGIN    TEXT LENGTH=8       ORIGIN STATION NAME
TYPE      DATA F'0'           ORIGIN STATION TYPE AND SUBTYPE
```

In example 1, a message is received from the queue of the station that has the name of the program. The message is placed in the buffer at TPBUF. The name of the station that originated the message is moved into the field ORIGIN, and its type and subtype are moved into the field at TYPE.

```
2         RECEIVE M,WASTEBUF,'$.WASTE',HEADER=HDR

WASTEBUF  DEFINE BUFFER,SIZE=256
HDR       DATA 12X'0000'        MESSAGE HEADER
```

In example 2, a message is received from the queue for station $.WASTE into a buffer at WASTEBUF. The header is moved into HDR.

RECEIVE N—Receive Notification of Messages

This instruction checks a station's message queue and notifies you (through the return code) of whether or not there is a message on the queue.

RECEIVE N Format

[*label*]	**RECEIVE N**	[,,*staname*][11] [,WAIT={<u>YES</u> | NO}] [,EXIT=*label*] [,P2=*name2*]

RECEIVE N Operands

Pn	Operand	Syntax	Description
P2	*staname*	location	The location of the 8-character name of the station whose queue is to be examined. The default is the name of the program that issues the instruction.
		'string'	The station name, enclosed in quotes.
	WAIT=	literal	An indicator of whether or not the instruction is to wait until there is a message on the queue. If the station is deleted while a wait is in effect, the instruction returns a return code of 2.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

RECEIVE N Return Codes

-2	There is a message on the disk queue, and no message on the storage queue.
-1	There is a message on the storage queue.
1	There is no message on the storage queue or the disk queue.
2	The station specified by *staname* does not exist.
4	Messages for *staname* are being held as a result of a CP F command that set output hold.

[11] The Communications Facility treats *staname* as the second positional operand (as it is in the other RECEIVE instructions). Therefore you must code two commas before *staname*. If you enter a value for the first positional operand, it is ignored.

RECEIVE N Example

```
RECEIVE NOTIFY,,'TPQUEUE',WAIT=NO
```

This example checks the queue of the station TPQUEUE and returns a return code indicating whether or not there is a message on the queue.

RECEIVE P—Purge a Message

This instruction purges a message that was previously received by a RECEIVE instruction with option COPY or KEEP. If the message to be purged exists both on disk and in storage as a result of a RECEIVE instruction with OPTION=KEEP, both copies of the message are purged. If the previous RECEIVE instruction against the queue did not specify OPTION=COPY or OPTION=KEEP, no message is purged.

RECEIVE P Format

[*label*]	**RECEIVE P**	**[,,***staname***]**[12] **[,WAIT={YES | NO}]** **[,EXIT=***label***]** **[,P2=***name2***]**

RECEIVE P Operands

Pn	Operand	Syntax	Description
P2	*staname*	location	The location of the 8-character name of the station from whose queue a message is to be purged. The default is the name of the program that issues the instruction.
		'string'	The station name, enclosed in quotes.
	WAIT=	literal	An indicator of whether or not the instruction is to wait until there is a message on the queue.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

RECEIVE P Return Codes

If the message to be purged was both on disk and in storage, the task code word contains the return code for the purge of the storage-queued message.

-1 Successful. The message was purged, or the previous RECEIVE instruction did not specify OPTION=COPY or OPTION=KEEP.

1 There is no message on the queue and WAIT=NO was specified.

2 The station specified by *staname* does not exist.

[12] The Communications Facility treats *staname* as the second positional operand (as it is in the other RECEIVE instructions). Therefore you must code two commas before *staname*. If you enter a value for the first positional operand, it is ignored.

3	There is an incorrect forward pointer in the storage queue. Report the problem to your IBM representative.
4	There is an incorrect backward pointer in the storage queue. Report the problem to your IBM representative.
7	A disk I/O error occurred; no message was purged.
8	The message to be purged is disk-queued and the station is not active; no message was purged.

RECEIVE P Example

```
RECEIVE PURGE,,'TPQUEUE',WAIT=NO
```

This example purges the first message on the queue for station TPQUEUE.

RECEIVE T—Receive a Message into a Text Area

The RECEIVE T instruction receives a message from a station's queue into an EDX text area. On successful completion, the message length is in the text header field (*text*-1).

RECEIVE T Format

[*label*]	**RECEIVE T**	,*text* [,*staname*] **[,WAIT={<u>YES</u> | NO}]** **[,ACK={<u>YES</u> | NO}]** **[,HEADER=***location***}** **{,ORIGIN=***location* **|]** **[,OPTION={COPY | KEEP}]** **[,EXIT=***label***]** **[,P1=***name1***]** **[,P2=***name2***]** **[,P3=***name3***]**

RECEIVE T Operands

Pn	**Operand**	**Syntax**	**Description**
P1	*text*	location	The address of the EDX text area into which the message is to be placed.
P2	*staname*	location	The location of the 8-character name of the station from whose queue the message is to be received. The default is the name of the program that issues the instruction.
		'string'	The station name, enclosed in quotes.
	WAIT=	literal	An indicator of whether or not the instruction is to wait until there is a message on the queue. If the station is deleted while a wait is in effect, the instruction returns a return code of 2.
	ACK=	literal	An indicator of whether or not receipt of the message is to be acknowledged to the sender of the message. ACK=YES has no effect if the sender is in a different node.

Pn	Operand	Syntax	Description
P3	HEADER=	location	The address at which the message header is to be placed. The area at *location* must be at least 24 bytes long to accommodate the header.
P3	ORIGIN=	location	The address at which the name of the station that originated the message is to be placed. The area at *location* must be at least 10 bytes long. Bytes 1 to 8 contain the station name, and bytes 9 and 10 contain the station type and subtype. If the origin of the message is not known, the area at *location* is filled in with binary zeros.
	OPTION=	literal	COPY indicates that a copy of the message is to be delivered normally, but the message is to be left on the queue. KEEP has the same effect as COPY, and also indicates that if the receiving program's buffer is too small to hold a disk-queued message, a copy of the message is to be kept in the system message buffer pool.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

RECEIVE T Return Codes

-19	The receive was completed successfully, and the disk-queue data set's capacity warning level, specified when the data set was defined, has been exceeded.
-18	The receive was completed successfully, and an overlay occurred in the disk-queue data set.
-1	Successful. A message has been moved to the text area at *text*.
1	There is no message on the queue and WAIT=NO was specified.
2	The station specified by *staname* does not exist.
3	The text area at *text* isn't big enough to contain the message; the message has been truncated. If OPTION=KEEP was specified, the text header count field contains the lesser of 255 and the message length; otherwise it contains the number of bytes moved to the text area.
4	Messages for *staname* are being held as a result of a CP F command that set output hold. No message was received.
5	The text area address specified was 0.

6 A status message (one sent with a SEND STATUS instruction) was received.

7 The message to be received was disk-queued and a disk I/O error occurred. No message was received.

8 The message to be received was disk-queued and the station specified in *staname* is not active. No message was received.

9 The previous receive from the queue specified by *staname* specified OPTION=COPY or OPTION=KEEP, and no RECEIVE P instruction has been issued. No message was received.

RECEIVE T Examples

```
1       RECEIVE TEXT,RECV,'STA1',ORIGIN=FROM

RECV     TEXT LENGTH=200
FROM     TEXT LENGTH=8          ORIGIN STATION NAME
TYPE     DATA F'0'              ORIGIN STATION TYPE AND SUBTYPE
```

In example 1, a message is received from the queue of the station STA1. The message is placed in the text area at RECV. The name of the station that originated the message is moved into the field FROM, and its type and subtype are moved into the field at TYPE.

```
2       RECEIVE T,TEXT,HEADER=HEAD

TEXT     TEXT LENGTH=100
HEAD     TEXT 12F'0'            MESSAGE HEADER
```

In example 2, a message is received from the queue associated with the station that has the same name as the program. The message is moved into the text area at TEXT. The header is moved into HEAD.

SEND A—Send Acknowledgment

This instruction sends an acknowledgment to a program that may be waiting at a SEND instruction with ACK=YES. This instruction allows the receiving program to receive a message, process it, and then acknowledge the receipt to the sending program. If the sender and receiver are in different nodes, the instruction has no effect.

SEND A Format

[*label*]	**SEND A**	,*staname* ,*code* **[,TYPE=NA]** **[,EXIT=***label***]** **[,P1=***name1***]** **[,P2=***name2***]**

SEND A Operands

Pn	**Operand**	**Syntax**	**Description**
P1	*staname*	location	The location of the 8-character name of the station that is the origin of the message whose receipt is being acknowledged. If TYPE=NA is coded, this must be the location of the word that contains the station's network address.
		'string'	The station name, enclosed in quotes. This syntax is not allowed with TYPE=NA.
P2	*code*	integer	The return code to be returned to the sender. The value must be in the range 20-32767. The value is made negative and placed in the sender's TCB code word.
	TYPE=	literal	NA if the *staname* operand specifies the station by its network address.
	EXIT=	label	The label of the next instruction to be executed the instruction completes with a positive return code.

SEND A Return Codes

-1	Successful.
1	The station specified by *staname* does not exist.
2	The station is not the origin of a message for which acknowledgment is pending.

SEND A Example

If one program sends a message with this instruction:

```
SEND M,'PROG2',MSGBUF,ORIGIN=MYSTA,ACK=YES
```

and another program receives it with this instruction:

```
RECEIVE M,INPUT,'PROG2',ORIGIN=FROMSTA,ACK=NO
```

then the second program can acknowledge receipt of the message with this instruction:

```
SEND ACK,FROMSTA,20
```

The first program's SEND instruction will complete with return code -20.

SEND CP—Send a Command

This instruction sends a CP or PD command from an EDX text area. The amount of data sent is determined by the text header field (*text*-1). Do not include the prefix CP in the message text. All the commands are explained in the *Operator's Guide.*

SEND CP Format

[*label*]	**SEND CP**	,[*staname*] ,*text* [{**,ORIGIN=***origin* | **,HEADER=***location*}] [**,PRI=**{<u>127</u> | *priority*}] [**,ACK=**{**YES** | <u>**NO**</u>}] [**,WAIT=**{<u>**YES**</u> | **NO**}] [**,EXIT=***label*] [**,P1=***name1*] [**,P2=***name2*] [**,P3=***name3*]

SEND CP Operands

Pn	Operand	Syntax	Description
P1	*staname*	location	The location of the 8-character name of any station in the node to which the command is to be sent; required only if the command is to be executed in a node other than the local node. If this operand is omitted, or specifies a station in the local node, the command is sent to the command processor in the local node.
		'string'	The station name, enclosed in quotes.
P2	*text*	location	The address of an EDX text area that contains the command to be sent.
		'string'	The actual command text; up to 254 characters, enclosed in quotes.
P3	ORIGIN=	location	The location of the name of the station that is the origin of this message. The default is the name of the sending program.

Pn	Operand	Syntax	Description
P3	**HEADER=**	location	The location at which the 24-byte message header is stored. If this operand is specified, *staname* and PRI are ignored; the message header must include the destination and priority. If this operand is not specified, the message dispatcher will build the message header.
	PRI=	integer	The priority of the command message, where 1 is the highest and 127 is the lowest.
	ACK=	literal	An indicator of whether or not this instruction is to wait for an acknowledgment from the command processor. ACK=YES is effective only when the message is sent to the command processor in the local node. If the message is sent to a remote node, receipt is acknowledged by the I/O control program that will transfer the message to the remote node.
	WAIT=	literal	YES if this instruction is to wait, if necessary, until storage is available in the message buffer pool to contain the command message.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

SEND CP Return Codes

-2 to -99 The negative value of the number of a CP error message issued when the command was executed. These conditions are returned only when ACK=YES. All the error messages are documented in the *Operator's Guide.*

-1 Successful. If ACK=NO, the message was sent to the command processor. If ACK=YES, the command was successfully executed by the local command processor or received by the I/O control program that will transfer the message to a remote node.

1 The message length is 0.

2 The destination station specified does not exist.

3 Storage is not available to hold the message and WAIT=NO was coded.

4 The origin station is prevented from sending messages as a result of a CP F command that set input hold.

SEND CP Examples

```
1         SEND CP,,CPSTART

2         SEND CP,'NODE2',CPSTART

CPSTART   TEXT 'S PROGRAM'
```

Example 1 sends a CP START command to the command processor, which starts the station called PROGRAM.

Example 2 specifies a destination station in a remote node. The command processor in that remote node starts a station called PROGRAM.

SEND E—Send an Error Message

SEND E sends a formatted error message to the Communications Facility system log. This instruction is identical to SEND L except that the default for the TYPE operand is E rather than I.

SEND E Format

[*label*]	**SEND E**	**,*number*** **[,*text*]** **[,COPY=*location*]** **[,ID={C'*id*' | <u>C'UM'</u>}]** **[,XCODE={*code* | <u>0000</u>}]** **[,TYPE={C | I | <u>E</u>}]** **[,P1=*name1*]** **[,P2=*name2*]** **[,P3=*name3*]** **[,P4=*name4*]** **[,P5=*name5*]**

SEND E Operands

Pn	**Operand**	**Syntax**	**Description**
P1	*number*	term	The number (1-99) of the message in a member of the $.SYSMSG data set. The member is specified in the ID= operand. If you specify a value other than 1-99, the message number field of the displayed message will contain **.
P2	*text*	location	The address of an EDX text area containing text you want to print in the text portion of the message. This text must not cause the entire message to exceed 72 characters. If it does, excess data is truncated.
		'string'	A literal string of message text, enclosed in quotes. This text must not cause the entire message to exceed 72 characters. If it does, excess data is truncated.
P3	**COPY=**	*location*	The address of an EDX text area where a copy of the message is to be placed. You can use the copy to send the message to a second destination.

Pn	Operand	Syntax	Description
P4	ID=	term	The 2-character name of the member of the $.SYSMSG data set where the message text is stored. This operand is used in conjunction with the *number* operand. If you specify a member that doesn't exist, the message as displayed contains no fixed text.
P5	XCODE=	integer	A value to be included in the message, displayed as 4 hexadecimal digits.
	TYPE=	literal	An EBCDIC character to be inserted into the message to indicate its type: E for error, I for informational, C for comment. If you code C, you must include the COPY= operand. TYPE=C together with COPY= creates a copy of the message without sending the message to the system log. SEND E also has two special type codes: X, which is used only in task error exit routines to log special data; and D, which produces a diagnostic dump. See "Using Task Error Exits" on page 29 and "Using Diagnostic Dumps" on page 29 for an explanation of these special cases.

SEND E Return Codes

-1 Successful.

SEND E Examples

1 SEND ERROR,22,'$.SPOOL',ID=C'AB'

If the text of message 22 in member AB of $.SYSMSG is "IS NOT DEFINED", and program PROGA issues this instruction, this message is sent to the system log:

```
*hh:mm:ss AB22 E 0000 PROGA $.SPOOL IS NOT DEFINED
```

2 SEND ERROR,10,DS1+$DSCBNAM,ID=C'AB',XCODE=DS1*

Assume that DS1 is a data set control block for data set MASTER02 and that this instruction is being used to report a disk I/O error. If the text of message AB10 is "DISK I/O ERROR", and the disk I/O return code is 10, this message is sent to the system log:

```
*hh:mm:ss AB10 E 000A PROGA MASTER02 DISK I/O ERROR
```

SEND L—Send a Log Message

SEND L sends a formatted log message to the Communications Facility system log. This instruction is identical to SEND E except that the default for the TYPE operand is I rather than E.

SEND L Format

[*label*]	**SEND L**	**,***number* **[,***text***]** **[,COPY=***location***]** **[,ID={C'***id***' | C'UM'}]** **[,XCODE={***code* **| 0000}]** **[,TYPE={C | E | I}]** **[,P1=***name1***]** **[,P2=***name2***]** **[,P3=***name3***]** **[,P4=***name4***]** **[,P5=***name5***]**

SEND L Operands

Pn	Operand	Syntax	Description
P1	*number*	term	The number (1-99) of the message in a member of the $.SYSMSG data set. The member is specified in the ID= operand. If you specify a value other than 1-99, the message number field of the displayed message will contain **.
P2	*text*	location	The address of an EDX text area containing text you want to print in the text portion of the message. This text must not cause the entire message to exceed 72 characters. If it does, excess data is truncated.
		'string'	A literal string of message text, enclosed in quotes. This text must not cause the entire message to exceed 72 characters. If it does, excess data is truncated.
P3	COPY=	*location*	The address of an EDX text area where a copy of the message is to be placed. You can use the copy to send the message to a second destination.

Pn	Operand	Syntax	Description
P4	ID=	term	The 2-character name of the member of the $.SYSMSG data set where the message text is stored. This operand is used in conjunction with the *number* operand. If you specify a member that doesn't exist, the message as displayed contains no fixed text.
P5	XCODE=	integer	A value to be included in the message, displayed as 4 hexadecimal digits.
	TYPE=	literal	An EBCDIC character to be inserted into the message to indicate its type: E for error, I for informational, C for comment. If you code C, you must include the COPY= operand. TYPE=C together with COPY= creates a copy of the message without sending the message to the system log.

SEND L Return Codes

-1 Successful.

SEND L Example

```
SEND LOG,22,'$.SPOOL',ID=C'AB'
```

If the text of message 22 in member AB of $.SYSMSG is "IS NOT DEFINED", and program PROGA issues this instruction, this message is sent to the system log:

```
hh:mm:ss AB22 I 0000 PROGA $.SPOOL IS NOT DEFINED
```

SEND M—Send a Message from a Buffer

This instruction sends a message to a station from a Communications Facility buffer. The amount of data sent is determined by the buffer header field B$COUNT.

SEND M Format

[*label*]	**SEND M**[13]	,[*staname*] ,*buffer* [{**,ORIGIN=***origin* | **,HEADER=***location*}] [**,PRI=**{<u>127</u> | *priority*}] [**,OPTION=**(*option1,option2...*)] [**,ACK=**{**YES** | <u>**NO**</u>}] [**,WAIT=**{<u>**YES**</u> | **NO**}] [**,EXIT=***label*] [**,P1=***name1*] [**,P2=***name2*] [**,P3=***name3*]

SEND M Operands

Pn	Operand	Syntax	Description
P1	*staname*	location	The location of the 8-character name of the destination station. This operand is required unless the originating station is linked to a destination or HEADER= is specified.
		'string'	The station name, enclosed in quotes.
P2	*buffer*	location	The address of the buffer that contains the message.
P3	ORIGIN=	location	The location of the name of the station that is the origin of this message. The default is the name of the sending program.
P3	HEADER=	location	The location at which the 24-byte message header is stored. If this operand is specified, *staname* and PRI are ignored; the message header must include the destination and priority. If this operand is not specified, the message dispatcher will build the message header.

13 Any characters *except* T can follow the M. For example, you can enter SEND MESSAGE or SEND MUSH, but not SEND MT or SEND MTEXT. This restriction applies because SEND MT is another instruction.

Pn	Operand	Syntax	Description
	PRI=	integer	The priority of the message, where 1 is the highest and 127 is the lowest.
	OPTION=	literal	WASTE if the message is to be sent to the station $.WASTE, overriding *staname* or the station to which the origin is linked. If $.WASTE is not active, the message is discarded. DISCARD if the message is to be discarded when it is undeliverable, instead of being sent to $.WASTE. NOPOST if this instruction is to wait only until the message has been placed in the message buffer pool, not until it has been placed on the destination queue.
	ACK=	literal	YES if this instruction is to wait for an acknowledgment from the receiving program; NO if no wait is to occur. ACK=YES is effective only for messages sent within the same node; if you code ACK=YES for a message sent to another node, receipt is acknowledged by the I/O control program that will transfer the message to the remote node. If the destination is a message station, ACK is ignored.
	WAIT=	literal	YES if this instruction is to wait, if necessary, until storage is available in the message buffer pool to contain the message. If the message is to be disk-queued, this operand does *not* cause the instruction to wait until there is enough space in the disk data set.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

SEND M Return Codes

-20 to -32767 User-assigned return codes, returned by the SEND ACK instruction; the send was successful.

-19 The send was successful, and the disk-queue data set's capacity warning level, specified when the data set was defined, has been exceeded.

-18 The send was successful, and an overlay occurred in the disk-queue data set.

-3	The message was sent with ACK=YES and the message was received, but it was truncated because the receiver's buffer or text area was too small. If the receiver is I/O control program $.IO0A10 or $.IO0AD0, the message is subsequently received successfully without truncation; return code -3 occurs because of the way these programs manage their buffers.
-1	Successful. If ACK=NO, the message was sent to the station's queue. If ACK=YES, the message was successfully received by its local destination or by the I/O control program that will transfer the message to a remote node.
1	The message length is 0.
2	The destination station specified does not exist, or, if the destination is in a different node, the path to that node does not exist.
3	Storage is not available to hold the message and WAIT=NO was coded.
4	The origin station is prevented from sending messages as a result of a CP F command that set input hold. No message was sent.
5	The message was to be disk-queued, and not enough space was available in the disk-queue data set. No message was sent.
6	The message was to be disk-queued and the message is longer than the disk-queue data set. No message was sent.
7	The message was to be disk-queued and a disk I/O error occurred. No message was sent.
8	The destination station is stopped and it is not a message station. No message was sent.

SEND M Example

```
         SEND MESSAGE,,TPBUF,ORIGIN=ORG
TPBUF    DEFINE BUFFER,SIZE=1024
ORG      TEXT 'ORGTERM'
```

In this example, a message in the buffer TPBUF is sent to the station to which the station ORGTERM is linked.

SEND MT—Send a Transaction from a Buffer

This instruction sends a transaction message to the program dispatcher from a Communications Facility buffer. The amount of data sent is determined by the buffer header field B$COUNT.

SEND MT Format

[*label*]	**SEND MT**	**,,***buffer*[14] **[,ORIGIN=***origin***]** **[,PRI={**<u>**127**</u> **|** *priority***}]** **[,OPTION=(***option1,option2...***)]** **[,ACK={YES |** <u>**NO**</u>**}]** **[,WAIT={**<u>**YES**</u> **| NO}]** **[,EXIT=***label***]** **[,P2=***name2***]** **[,P3=***name3***]**

SEND MT Operands

Pn	Operand	Syntax	Description
P2	*buffer*	location	The address of the buffer that contains the transaction message.
P3	ORIGIN=	location	The location of the name of the station that is the origin of this transaction message. The default is the name of the sending program.
	PRI=	integer	The priority of the transaction message, where 1 is the highest and 127 is the lowest.
	OPTION=	literal	WASTE if the message is to be sent to the station $.WASTE rather than to the program dispatcher. If $.WASTE is not active, the message is discarded. DISCARD if the message is to be discarded when it is undeliverable, instead of being sent to $.WASTE.

14 The Communications Facility treats *buffer* as the second positional operand (as it is in the other SEND instructions). Therefore you must code two commas before *buffer*. If you enter a value for the first positional operand, it is ignored.

Pn	Operand	Syntax	Description
			NOPOST if this instruction is to wait only until the message has been placed in the message buffer pool, not until it has been placed on the destination queue.
	ACK=	literal	YES if this instruction is to wait for an acknowledgment from the program dispatcher; NO if no wait is to occur.
	WAIT=	literal	YES if this instruction is to wait, if necessary, until storage is available in the message buffer pool to contain the transaction message. If the message is to be disk-queued, this operand does *not* cause the instruction to wait until there is enough space in the disk data set.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

SEND MT Return Codes

-20	to -32767 User-assigned return codes, returned by the SEND ACK instruction; the send was successful.
-19	The send was successful, and the disk-queue data set's capacity warning level, specified when the data set was defined, has been exceeded.
-18	The send was successful, and an overlay occurred in the disk-queue data set.
-3	The message was sent with ACK=YES and the message was received, but it was truncated because the receiver's buffer or text area was too small. If the receiver is I/O control program $.IO0A10 or $.IO0AD0, the message is subsequently received successfully without truncation; return code -3 occurs because of the way these programs manage their buffers.
-1	Successful. If ACK=NO, the message was sent to the program dispatcher. If ACK=YES, the message was successfully received by the program dispatcher.
1	The message length is 0.
2	The program dispatcher's user station does not exist.
3	Storage is not available to hold the message and WAIT=NO was coded.
4	The origin station is prevented from sending messages as a result of a CP F command that set input hold. No message was sent.
5	The message was to be disk-queued, and not enough space was available in the disk-queue data set. No message was sent.

6 The message was to be disk-queued and the message is longer than the disk-queue data set. No message was sent.

7 The message was to be disk-queued and a disk I/O error occurred. No message was sent.

8 The program dispatcher is stopped. No message was sent.

SEND MT Example

```
                PUT FIELD,TRANS,SCHDTRNS,OPTION=INITIAL
                SEND MT,,TRANS
TRANS           DEFINE BUFFER,SIZE=512
SCHDTRNS        TEXT 'SCHDHSLGITHS3020000'
```

In this example, a transaction message in the buffer TRANS is sent to the program dispatcher. The transaction ID is SCHD, and the cell ID is HS.

SEND S—Send a Status Message from an EDX Text Area

This instruction sends a status message to a station from an EDX text area. The amount of data sent is determined by the text header field (*text*-1).

SEND S Format

[*label*]	**SEND S**[15]	[,*staname*] ,*text* [,**ORIGIN**=*origin*] [,**PRI**={<u>**127**</u> | *priority*}] [,**OPTION**=(*option1,option2...*)] [,**ACK**={**YES** | <u>**NO**</u>}] [,**WAIT**={<u>**YES**</u> | **NO**}] [,**EXIT**=*label*] [,**P1**=*name1*] [,**P2**=*name2*] [,**P3**=*name3*]

SEND S Operands

Pn	Operand	Syntax	Description
P1	*staname*	location	The location containing the 8-character name of the destination station. This operand is required unless the originating station is linked to a destination.
		'string'	The station name, enclosed in quotes.
P2	*text*	location	The location of the text area that contains the status message.
		'string'	A string of up to 254 characters, enclosed in quotes.
P3	**ORIGIN=**	location	The location of the name of the station that is the origin of this message. The default is the name of the sending program.
	PRI=	integer	The priority of the message, where 1 is the highest and 127 is the lowest.

15 Any characters *except* M can follow the S. This restriction applies because SEND SM is another instruction.

Pn	Operand	Syntax	Description
	OPTION=	literal	WASTE if the message is to be sent to the station $.WASTE, overriding *staname* or the station to which the origin is linked. If $.WASTE is not active, the message is discarded. DISCARD if the message is to be discarded when it is undeliverable, instead of being sent to $.WASTE. NOPOST if this instruction is to wait only until the message has been placed in the message buffer pool, not until it has been placed on the destination queue.
	ACK=	literal	YES if this instruction is to wait for an acknowledgment from the receiving program; NO if no wait is to occur. ACK=YES is valid only for messages sent within the same node. If the destination is a message station, ACK is ignored.
	WAIT=	literal	YES if this instruction is to wait, if necessary, until storage is available in the system buffer to contain the message. This operand causes the instruction to wait only until there is enough storage to send the message. If the message is to be disk-queued, this operand does *not* cause the instruction to wait until there is enough space in the disk data set.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

SEND S Return Codes

-20	to -32767 User-assigned return codes, returned by the SEND ACK instruction; the send was successful.
-19	The send was successful, and the disk-queue data set's capacity warning level, specified when the data set was defined, has been exceeded.
-18	The send was successful, and an overlay occurred in the disk-queue data set.
-3	The message was sent with ACK=YES and the message was received, but it was truncated because the receiver's buffer or text area was too small. If the receiver is I/O control program $.IO0A10 or $.IO0AD0, the message is subsequently received successfully without truncation; return code -3 occurs because of the way these programs manage their buffers.

-1	Successful. If ACK=NO, the message was sent to the station's queue. If ACK=YES, the message was successfully received by its local destination or by the I/O control program that will transfer the message to a remote node.
1	The message length is 0.
2	The destination station specified does not exist, or, if the destination is in a different node, the path to that node does not exist.
3	Storage is not available to hold the message and WAIT=NO was coded.
4	The origin station is prevented from sending messages as a result of a CP F command that set input hold. No message was sent.
5	The message was to be disk-queued, and not enough space was available in the disk-queue data set. No message was sent.
6	The message was to be disk-queued and the message is longer than the disk-queue data set. No message was sent.
7	The message was to be disk-queued and a disk I/O error occurred. No message was sent.

SEND S Example

```
SEND STATUS,'PROGA','P'
```

This example sends a status message to the station PROGA to tell the program to stop.

SEND SM—Send a Status Message from a Buffer

This instruction sends a status message to a station from a Communications Facility buffer. The amount of data sent is determined by the buffer header field B$COUNT.

SEND SM Format

[*label*]	**SEND SM**	[,*staname*] ,*buffer* [,**ORIGIN=***origin*] [,**PRI=**{<u>127</u> | *priority*}] [,**OPTION=**(*option1,option2...*)] [,**ACK=**{YES | <u>NO</u>}] [,**WAIT=**{<u>YES</u> | NO}] [,**EXIT=***label*] [,**P1=***name1*] [,**P2=***name2*] [,**P3=***name3*]

SEND SM Operands

Pn	Operand	Syntax	Description
P1	*staname*	location	The location containing the 8-character name of the destination station. This operand is required unless the originating station is linked to a destination.
		'string'	The station name, enclosed in quotes.
P2	*buffer*	location	The address of the buffer that contains the status message.
P3	**ORIGIN=**	location	The location of the name of the station that is the origin of this message. The default is the name of the sending program.
	PRI=	integer	The priority of the message, where 1 is the highest and 127 is the lowest.
	OPTION=	literal	WASTE if the message is to be sent to the station $.WASTE, overriding *staname* or the station to which the origin is linked. If $.WASTE is not active, the message is discarded.

Pn	Operand	Syntax	Description
			DISCARD if the message is to be discarded when it is undeliverable, instead of being sent to $.WASTE. NOPOST if this instruction is to wait only until the message has been placed in the message buffer pool, not until it has been placed on the destination queue.
	ACK=	literal	YES if this instruction is to wait for an acknowledgment from the receiving program; NO if no wait is to occur. ACK=YES is valid only for messages sent within the same node. If the destination is a message station, ACK is ignored.
	WAIT=	literal	YES if this instruction is to wait, if necessary, until storage is available in the system buffer to contain the message. This operand causes the instruction to wait only until there is enough storage to send the message. If the message is to be disk-queued, this operand does *not* cause the instruction to wait until there is enough space in the disk data set.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

SEND SM Return Codes

-20 to -32767 User-assigned return codes, returned by the SEND ACK instruction; the send was successful.

-19 The send was successful, and the disk-queue data set's capacity warning level, specified when the data set was defined, has been exceeded.

-18 The send was successful, and an overlay occurred in the disk-queue data set.

-3 The message was sent with ACK=YES and the message was received, but it was truncated because the receiver's buffer or text area was too small. If the receiver is I/O control program $.IO0A10 or $.IO0AD0, the message is subsequently received successfully without truncation; return code -3 occurs because of the way these programs manage their buffers.

-1 Successful. If ACK=NO, the message was sent to the station's queue. If ACK=YES, the message was successfully received by its local destination or by the I/O control program that will transfer the message to a remote node.

1 The message length is 0.

2 The destination station specified does not exist, or, if the destination is in a different node, the path to that node does not exist.

3 Storage is not available to hold the message and WAIT=NO was coded.

4 The origin station is prevented from sending messages as a result of a CP F command that set input hold. No message was sent.

5 The message was to be disk-queued, and not enough space was available in the disk-queue data set. No message was sent.

6 The message was to be disk-queued and the message is longer than the disk-queue data set. No message was sent.

7 The message was to be disk-queued and a disk I/O error occurred. No message was sent.

SEND SM Example

```
SEND SM,'PROGA',OUTBUFF
OUTBUFF DEFINE BUFFER,SIZE=512
```

This example sends the status message in buffer OUTBUFF to station PROGA.

SEND T—Send a Message from an EDX Text Area

This instruction sends a message to a station from an EDX text area. The amount of data sent is determined by the text header field (*text*-1).

SEND T Format

[*label*]	**SEND T**[16]	**,**[*staname*] **,***text* [**,ORIGIN=***origin*] [**,PRI=**{**127** | *priority*}] [**,OPTION=**(*option1,option2...*)] [**,ACK=**{**YES** | **NO**}] [**,WAIT=**{**YES** | **NO**}] [**,EXIT=***label*] [**,P1=***name1*] [**,P2=***name2*] [**,P3=***name3*]

SEND T Operands

Pn	Operand	Syntax	Description
P1	*staname*	location	The location of the 8-character name of the destination station. This operand is required unless the originating station is linked to a destination.
		'string'	The station name, enclosed in quotes.
P2	*text*	location	The address of the text area that contains the message.
		'string'	A string of up to 254 characters, enclosed in quotes.
P3	ORIGIN=	location	The location of the name of the station that is the origin of this message. The default is the name of the sending program.
	PRI=	integer	The priority of the message, where 1 is the highest and 127 is the lowest.

16 Any characters *except* T can follow the T. For example, you can enter SEND TEXT or SEND TIP, but not SEND TT or SEND TTEXT. This restriction applies because SEND TT is another instruction.

Pn	Operand	Syntax	Description
	OPTION=	literal	WASTE if the message is to be sent to the station $.WASTE, overriding *staname* or the station to which the origin is linked. If $.WASTE is not active, the message is discarded. DISCARD if the message is to be discarded when it is undeliverable, instead of being sent to $.WASTE. NOPOST if this instruction is to wait only until the message has been placed in the message buffer pool, not until it has been placed on the destination queue.
	ACK=	literal	YES if this instruction is to wait for an acknowledgment from the receiving program; NO if no wait is to occur. ACK=YES is effective only for messages sent within the same node; if you code ACK=YES for a message sent to another node, receipt is acknowledged by the I/O control program that will transfer the message to the remote node. If the destination is a message station, ACK is ignored.
	WAIT=	literal	YES if this instruction is to wait, if necessary, until storage is available in the message buffer pool to contain the message. If the message is to be disk-queued, this operand does *not* cause the instruction to wait until there is enough space in the disk data set.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

SEND T Return Codes

-20 to -32767 User-assigned return codes, returned by the SEND ACK instruction; the send was successful.

-19 The send was successful, and the disk-queue data set's capacity warning level, specified when the data set was defined, has been exceeded.

-18 The send was successful, and an overlay occurred in the disk-queued data set.

-3 The message was sent with ACK=YES and the message was received, but it was truncated because the receiver's buffer or text area was too small. If the receiver is I/O control program $.IO0A10 or $.IO0AD0, the message is subsequently received successfully without truncation; return code -3 occurs because of the way these programs manage their buffers.

-1	Successful. If ACK=NO, the message was sent to the station's queue. If ACK=YES, the message was successfully received by its local destination or by the I/O control program that will transfer the message to a remote node.
1	The message length is 0.
2	The destination station specified does not exist, or, if the destination is in another node, the path to that node does not exist.
3	Storage is not available to hold the message and WAIT=NO was coded.
4	The origin station is prevented from sending messages as a result of a CP F command that set input hold. No message was sent.
5	The message was to be disk-queued, and not enough space was available in the disk-queue data set. No message was sent.
6	The message was to be disk-queued and the message is longer than the disk-queue data set. No message was sent.
7	The message was to be disk-queued and a disk I/O error occurred. No message was sent.
8	The destination station is stopped and it is not a message station. No message was sent.

SEND T Example

```
         SEND TEXT,'PROGA',AREA
AREA     TEXT 'TEST MESSAGE'
```

In this example, the message 'TEST MESSAGE' is sent to the station PROGA.

SEND TT—Send a Transaction from an EDX Text Area

This instruction sends a transaction message to the program dispatcher from an EDX text area. The amount of data sent is determined by the text header field (*text*-1).

SEND TT Format

[*label*]	**SEND TT**	**,,***text*[17] **[,ORIGIN=***origin***]** **[,PRI={**<u>**127**</u> **|** *priority***}]** **[,OPTION=(***option1,option2...***)]** **[,ACK={YES |** <u>**NO**</u>**}]** **[,WAIT={**<u>**YES**</u> **| NO}]** **[,EXIT=***label***]** **[,P2=***name2***]** **[,P3=***name3***]**

SEND TT Operands

Pn	**Operand**	**Syntax**	**Description**
P2	*text*	location	The address of the EDX text area that contains the transaction message.
		'string'	A string of up to 254 characters, enclosed in quotes.
P3	**ORIGIN=**	location	The location of the name of the station that is the origin of this transaction message. The default is the name of the sending program.
	PRI=	integer	The priority of the transaction message, where 1 is the highest and 127 is the lowest.
	OPTION=	literal	WASTE if the message is to be sent to the station $.WASTE rather than to the program dispatcher. If $.WASTE is not active, the message is discarded. DISCARD if the message is to be discarded when it is undeliverable, instead of being sent to $.WASTE.

17 The Communications Facility treats *text* as the second positional operand (as it is in the other SEND instructions). Therefore you must code two commas before *text*. If you enter a value for the first positional operand, it is ignored.

Pn	Operand	Syntax	Description
			NOPOST if this instruction is to wait only until the message has been placed in the message buffer pool, not until it has been placed on the destination queue.
	ACK=	literal	YES if this instruction is to wait for an acknowledgment from the program dispatcher; NO if no wait is to occur.
	WAIT=	literal	YES if this instruction is to wait, if necessary, until storage is available in the message buffer pool to contain the transaction message. If the message is to be disk-queued, this operand does *not* cause the instruction to wait until there is enough space in the disk data set.
	EXIT=	label	The label of the next instruction to be executed if the instruction completes with a positive return code.

SEND TT Return Codes

-20	to -32767 User-assigned return codes, returned by the SEND ACK instruction; the send was successful.
-19	The send was successful, and the disk-queue data set's capacity warning level, specified when the data set was defined, has been exceeded.
-18	The send was successful, and an overlay occurred in the disk-queue data set.
-3	The message was sent with ACK=YES and the message was received, but it was truncated because the receiver's buffer or text area was too small. If the receiver is I/O control program $.IO0A10 or $.IO0AD0, the message is subsequently received successfully without truncation; return code -3 occurs because of the way these programs manage their buffers.
-1	Successful. If ACK=NO, the message was sent to the program dispatcher. If ACK=YES, the program dispatcher successfully received the message.
1	The message length is 0.
2	The program dispatcher's user station does not exist.
3	Storage is not available to hold the message and WAIT=NO was coded.
4	The origin station is prevented from sending messages as a result of a CP F command that set input hold. No message was sent.
5	The message was to be disk-queued, and not enough space was available in the disk-queue data set. No message was sent.

6 The message was to be disk-queued and the message is longer than the disk-queue data set. No message was sent.

7 The message was to be disk-queued and a disk I/O error occurred. No message was sent.

8 The program dispatcher is stopped. No message was sent.

SEND TT Example

```
SEND TT,,'SCHDHSLGITHS3020000'
```

In this example, the transaction SCHD is sent to the program dispatcher, which routes it to cell HS.

Coding Work Session Controller Transactions

This chapter shows the format of the transactions you use to communicate with the work session controller (WSC). It shows the format of each WSC transaction command and the corresponding acknowledgment transaction.

Transaction data is characters unless a specific transaction description specifies otherwise. In the transaction examples, the character "b" represents a blank.

Figure 33 summarizes which commands are valid for which terminals, and which ones require an acknowledgment. A static screen terminal is a 4978, 4979, 4980, or 3101 in block mode. A roll screen terminal is a 3101 in character mode. An output only terminal is a printer.

Transaction Command	Static Screen Terminal	Roll Screen Terminal	Output Only Terminal	Acknow-ledgment Required
BI–Send a Screen Image	Y	N	N	N
CC–Carriage Control	N	Y	Y	N
CD–Clear Data	Y	N	N	N
ES–End Session	Y	Y	Y	N
FT–Read Field Table	Y	N	N	Y
IT–Set Input Timer	Y	N	N	N
LI–Link to Another Program	Y	Y	Y	Y
LK–Lock Keyboard	Y	N	N	N
LS–Set Lock Sequence	Y	N	N	N
PD–Stop Device	Y	Y	Y	N
PW–Priority Write	Y	N	N	N
RA–Read all Data	Y	Y	N	Y
RC–Read Cursor	Y	N	N	Y
RD–Read Unprotected Data	Y	Y	N	Y
RS–Restore Data	Y	Y	Y	Y
RT–Read Program Function Key Table	Y	N	N	Y
SC–Set Cursor	Y	N	N	N

Figure 33 (Part 1 of 2). Work Session Controller Transaction Commands

Transaction Command	Static Screen Terminal	Roll Screen Terminal	Output Only Terminal	Acknow-ledgment Required
SD–Start Device	Y	Y	Y	N
SF–Set Forms	N	Y	Y	N
SL–Set Station Name	Y	Y	Y	N
SS–Start Session	Y	Y	Y	N
ST–Set Program Function Key Table	Y	N	N	N
SV–Save Data	Y	Y	Y	Y
TN–Sound Tone	Y	Y	N	N
UK–Unlock Keyboard	Y	N	N	N
US–End Lock Sequence	Y	N	N	N
WD–Write Unprotected Data	Y	Y	Y	N
WK–Wait for Key	Y	N	N	Y
WP–Write Protected Data	Y	N	N	N

Figure 33 (Part 2 of 2). Work Session Controller Transaction Commands

BI—Send a Screen Image

The BI command fetches a screen image from $.WSCIMG and displays it on the terminal.

BI Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22	23-30
WSC	*pc*	*sect*	*sc*	BI	*terminal*	*image*

WSC
is the primary transaction identifier.

pc
is the primary cell identifier (the cell in which the command is to be executed).

sect
is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

BI
is the work session controller command.

terminal
is the terminal name used in this session.

image
is the name of the screen image.

BI Acknowledgment Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
prit	*pc*	WSC	*sc*	BI	*terminal*

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

BI
is the work session controller command.

terminal
is the terminal name used in this session.

BI Example

To display the screen image SCRNIMAG on terminal 4978TERM:

WSCbbbPROGbbBI4978TERMSCRNIMAG

The acknowledgment transaction is:

PROGbbWSCb??BI4798TERM

indicating that the screen image SCRNIMAG was successfully displayed on terminal 4978TERM.

CC—Carriage Control

The CC command allows you to control the carriage of a roll screen or output only terminal.

CC Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22	23-24	25-26	27-28
WSC	*pc*	*sect*	*sc*	CC	*terminal*	*nn*	*ls*	*ss*

WSC
is the primary transaction identifier.

pc
is the primary cell identifier (the cell in which the command is to be executed).

sect
is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

CC
is the work session controller command.

terminal
is the terminal name used in this session.

nn
is the number of the line to go to. If this field is blank, it is ignored.

ls
is the number of lines to skip. If this field is blank or 0, it is ignored.

ss
is the number of spaces to skip. If this field is blank or 0, it is ignored.

CC Acknowledgment Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
prit	*pc*	WSC	*sc*	CC	*terminal*

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

CC
is the work session controller command.

terminal
is the terminal name used in this session.

CC Example

To go to the top of a form and space over 20 positions:

WSCbbbbbbbbCCSYSTPTRb00bb20

No acknowledgment is sent, because the secondary transaction identifier is blank. The roll screen terminal is at line 0, space 20.

CD—Clear Data

The CD command sets the unprotected data areas on a terminal to nulls.

CD Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
WSC	*pc*	*sect*	*sc*	CD	*terminal*

WSC
  is the primary transaction identifier.

pc
  is the primary cell identifier (the cell in which the command is to be executed).

sect
  is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
  is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

CD
  is the work session controller command.

terminal
  is the terminal name used in this session.

CD Acknowledgment Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
prit	*pc*	WSC	*sc*	CD	*terminal*

prit
  is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
  is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
  is the secondary transaction identifier.

sc
  is the secondary cell identifier (the cell where your original WSC transaction was executed).

CD
is the work session controller command.

terminal
is the terminal name used in this session.

CD Example

To clear the unprotected data areas on terminal 4978TERM:

WSCbbbMEN1bbCD4978TERM

The acknowledgment transaction is:

MEN1bbWSCb??CD4978TERM

All unprotected data areas are set to nulls.

ES—End a Session

The ES command ends a session with a terminal. The work session controller executes a DEQT for the EDX terminal whose name you supplied in the SD command.

ES Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
WSC	*pc*	*sect*	*sc*	ES	*terminal*

WSC
: is the primary transaction identifier.

pc
: is the primary cell identifier (the cell in which the command is to be executed).

sect
: is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
: is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

ES
: is the work session controller command.

terminal
: is the terminal name used in this session.

ES Acknowledgment Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
prit	*pc*	WSC	*sc*	ES	*terminal*

prit
: is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
: is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
: is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

ES
is the work session controller command.

terminal
is the terminal name used in this session.

ES Example

To end a session with the terminal 4978TERM:

WSCbbbPROGbbES4978TERM

The acknowledgment transaction is:

PROGbbWSCb??ES4978TERM

The EDX terminal referred to in this session as 4978TERM is released.

FT—Read the Field Table

The FT command reads the field table associated with a screen image in $.WSCIMG. The format of the field table is the same as for the $IMPROT subroutine, described in the *Communications and Terminal Applications Guide*. You can request that the field table be transferred to a buffer you specify or returned in the acknowledgment transaction. If you request that it be transferred to a buffer, the command can be executed only in the cell in which it is entered.

FT Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22	23-24	25-26	27-34
WSC	*pc*	*sect*	*sc*	FT	*terminal*	*ad*	*ak*	*image*

WSC
: is the primary transaction identifier.

pc
: is the primary cell identifier (the cell in which the command is to be executed).

sect
: is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
: is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

FT
: is the work session controller command.

terminal
: is the terminal name used in this session.

ad
: is the address, in binary, of the buffer where the field table is to be stored. Specify 0 if the field table is to be returned in the acknowledgment transaction. The buffer must be defined with an EDX BUFFER instruction.

ak
: is the address key associated with the buffer address, in binary. Specify -1 (X'FFFF') if the field table is to be returned in the acknowledgment transaction. (See the TCBGET instruction in the *EDX Language Reference* manual.)

image
: is the name of the screen image.

FT Acknowledgment Transaction Format (Required)

1-4	5-6	7-10	11-12	13-14	15-22	23-24	25-*n*
prit	*pc*	WSC	*sc*	FT	*terminal*	*rc*	*ftab*

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

FT
is the work session controller command.

terminal
is the terminal name used in this session.

rc
is a return code:

Y0—Operation was successful.
E1—Incorrect transaction length.
E2—Key specified is invalid.
E4—Data set member not defined.
E5—Disk I/O error occurred.
E6—Buffer too small.
E7—Request is not for this cell.

ftab
is the field table. This field is optional.

FT Examples

1 To read the field table associated with the screen image MENUSCRN into the buffer BUFF1 in partition 3:

BUFF1 BUFFER 100,BYTES

WSCbbbPROGbbFT4978TERM*xxyy*MENUSCRN

where *xx* is the address of BUFF1 and *yy* is X'0002' (the address key is 1 less than the partition number).

The acknowledgment transaction is:

PROGbbWSCb??FT4978TERMY0

The operation was successful and the field table is in BUFF1 in partition 3.

2 To receive the field table associated with the screen image MENUSCRN in the acknowledgment transaction:

WSCbbbPROGbbFT4978TERM*xxyy*MENUSCRN

where *xx* is X'0000' and *yy* is X'FFFF'.

The acknowledgment transaction is:

PROGbbWSCb??FT4978TERMY0*nnssr1c1s1r2c2s2*. . .

where:

nn
is the number of screen fields in the table.

ss
is the screen size.

rn
is the row position of field *n*.

cn
is the column position of field *n*.

sn
is the size of field *n*.

IT—Set Input Timer

The IT command sets a timer for a static screen terminal. Subsequent WK commands will time out when the time interval specified through IT has elapsed.

IT Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22	23-28
WSC	*pc*	*sect*	*sc*	IT	*terminal*	*hhmmss*

WSC
is the primary transaction identifier.

pc
is the primary cell identifier (the cell in which the command is to be executed).

sect
is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

IT
is the work session controller command.

terminal
is the terminal name used in this session.

hhmmss
is the time interval, in the form hours, minutes, seconds. Zero resets the timer.

IT Acknowledgment Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
prit	*pc*	WSC	*sc*	IT	*terminal*

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

IT
is the work session controller command.

terminal
is the terminal name used in this session.

IT Example

To set an input timer on terminal 4978TERM with a value of 2 minutes and 5 seconds:

WSCbbbPROGbbIT4978TERM000205

The acknowledgment transaction is:

PROGbbWSCb??IT4978TERM

The input timer has been set; 2 minutes and 5 seconds will be the timeout value used in subsequent WK transaction commands.

LI—Link to Another Program

The LI command links to another program and, optionally, passes data to it. The program it links to is the one associated with the secondary transaction identifier you specify. Note that you can achieve such linkage with any WSC transaction by specifying, as the secondary transaction identifier, the transaction associated with the program you want.

LI Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22	23-*n*
WSC	*pc*	*sect*	*sc*	LI	*terminal*	*data*

WSC
: is the primary transaction identifier.

pc
: is the primary cell identifier (the cell in which the command is to be executed).

sect
: is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
: is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

LI
: is the work session controller command.

terminal
: is the terminal name used in this session.

data
: is binary or character data to be passed to the program, 1-1920 bytes. This field is optional.

LI Acknowledgment Transaction Format (Required)

1-4	5-6	7-10	11-12	13-14	15-22	23-*n*
prit	*pc*	WSC	*sc*	LI	*terminal*	*data*

prit
: is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

LI
is the work session controller command.

terminal
is the terminal name used in this session.

data
is binary or character data passed by the program that issued the LI command, 1-1920 characters. This field is optional.

LI Example

To link to the program that is associated with transaction identifier USER:

WSCbbbUSERbbLI4978TERM

The acknowledgment transaction is:

USERbbWSCb??LI4978TERM

The acknowledgment is sent to the program associated with transaction identifier USER.

LK—Lock the Keyboard

The LK command locks the keyboard on a static screen terminal.

LK Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
WSC	*pc*	*sect*	*sc*	LK	*terminal*

WSC
is the primary transaction identifier.

pc
is the primary cell identifier (the cell in which the command is to be executed).

sect
is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

LK
is the work session controller command.

terminal
is the terminal name used in this session.

LK Acknowledgment Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
prit	*pc*	WSC	*sc*	LK	*terminal*

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

LK
: is the work session controller command.

terminal
: is the terminal name used in this session.

LK Example

To lock the keyboard of terminal 4978TERM:

WSCbbbMEN1bbLK4978TERM

The acknowledgment transaction is:

MEN1bbWSCb??LK4978TERM

The keyboard of terminal 4978TERM is locked.

LS—Set Lock Sequence

The LS command sets a lock sequence for a static screen terminal. When a lock sequence is in effect, a locked keyboard remains locked until a US command is issued. When a lock sequence is not in effect, a locked keyboard is unlocked when a WK command is issued and relocked when the wait key operation completes.

Use the US command to end a lock sequence.

LS Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
WSC	*pc*	*sect*	*sc*	LS	*terminal*

WSC
is the primary transaction identifier.

pc
is the primary cell identifier (the cell in which the command is to be executed).

sect
is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

LS
is the work session controller command.

terminal
is the terminal name used in this session.

LS Acknowledgment Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
prit	*pc*	WSC	*sc*	LS	*terminal*

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

LS
is the work session controller command.

terminal
is the terminal name used in this session.

LS Example

To set a lock sequence for terminal 4978TERM:

WSCbbbMEN1bbLS4978TERM

The acknowledgment transaction is:

MEN1bbWSCb??LS4978TERM

indicating that the lock sequence is set.

PD—Stop Device

The PD command stops a work session controller device and returns it to EDX control. It deletes the station block that the SD command created.

PD Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
WSC	*pc*	*sect*	*sc*	PD	*terminal*

WSC
is the primary transaction identifier.

pc
is the primary cell identifier (the cell in which the command is to be executed).

sect
is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

PD
is the work session controller command.

terminal
is the terminal name used in this session.

PD Acknowledgment Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
prit	*pc*	WSC	*sc*	PD	*terminal*

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

PD
is the work session controller command.

terminal
is the terminal name used in this session.

PD Example

To stop the work session controller device 4978TERM:

WSCbbbbbbbbbPD4978TERM

In this example, no acknowledgment is sent because the secondary transaction identifier is blank. The terminal 4978TERM is returned to EDX control, and the station block 4978TERM is deleted.

PW—Priority Write

The PW command writes data to the priority area (rows 23 and 24) of a static screen terminal.

PW Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22	23-24	25-26	27-*n*
WSC	*pc*	*sect*	*sc*	PW	*terminal*	*rr*	*cc*	*data*

WSC
is the primary transaction identifier.

pc
is the primary cell identifier (the cell in which the command is to be executed).

sect
is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

PW
is the work session controller command.

terminal
is the terminal name used in this session.

rr
is the beginning row in the priority area to be written.

cc
is the beginning column in the priority area to be written.

data
is the data to be written.

PW Acknowledgment Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
prit	*pc*	WSC	*sc*	PW	*terminal*

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
  is the secondary transaction identifier.

sc
  is the secondary cell identifier (the cell where your original WSC transaction was executed).

PW
  is the work session controller command.

terminal
  is the terminal name used in this session.

PW Example

To write data to the terminal 4978TERM starting at row 23, column 0:

WSCbbbPROGbbPW4978TERM2300ERROR ON INPUT

The acknowledgment transaction is:

PROGbbWSCb??PW4978TERM

The message is written to the priority area of the terminal.

RA—Read All Data

The RA command reads a specified number of characters of protected and unprotected data from a terminal, beginning at a specified location.

RA Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22	23-24	25-26	27-30
WSC	*pc*	*sect*	*sc*	RA	*terminal*	*rr*	*cc*	*numb*

WSC
is the primary transaction identifier.

pc
is the primary cell identifier (the cell in which the command is to be executed).

sect
is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

RA
is the work session controller command.

terminal
is the terminal name used in this session.

rr
is the beginning row to be read. For roll screen terminals, specify 00.

cc
is the beginning column to be read. For roll screen terminals, specify 00.

numb
is the number of characters to be read. For static screen terminals, *numb* can be from 1 to 1920; for roll screen terminals, it can be from 1 to 102.

RA Acknowledgment Transaction Format (Required)

1-4	5-6	7-10	11-12	13-14	15-22	23-24	25-26	27-30	31-*n*
prit	*pc*	WSC	*sc*	RA	*terminal*	*rr*	*cc*	*numb*	*data*

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

RA
is the work session controller command.

terminal
is the terminal name used in this session.

rr
is the row where the cursor is positioned (00 for roll screen terminals).

cc
is the column where the cursor is positioned (00 for roll screen terminals).

numb
is the number of characters read.

data
is the data read.

RA Example

To read 10 protected and unprotected characters from a terminal, TERM01, starting at row 22, column 60:

WSCbbbPROGbbRATERM01bb22600010

The acknowledgment transaction is:

PROGbbWSCb??RATERM01bb01000010MESSAGEb16

The cursor was at row 1, column 0 when the 10 characters (MESSAGEb16) were read.

RC—Read the Cursor

The RC command reads the cursor position from a static screen terminal.

RC Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
WSC	*pc*	*sect*	*sc*	RC	*terminal*

WSC
: is the primary transaction identifier.

pc
: is the primary cell identifier (the cell in which the command is to be executed).

sect
: is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
: is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

RC
: is the work session controller command.

terminal
: is the terminal name used in this session.

RC Acknowledgment Transaction Format (Required)

1-4	5-6	7-10	11-12	13-14	15-22	23-24	25-26
prit	*pc*	WSC	*sc*	RC	*terminal*	*rr*	*cc*

prit
: is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
: is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
: is the secondary transaction identifier.

sc
: is the secondary cell identifier (the cell where your original WSC transaction was executed).

RC
is the work session controller command.

terminal
is the terminal name used in this session.

rr
is the row position of the cursor.

cc
is the column position of the cursor.

RC Example

To read the cursor position on terminal 4979T1:

WSCb01MENAbbRC4979T1bb

The acknowledgment transaction is:

MENAbbWSCb01RC4979T1bb1001

The cursor is at row 10, column 1.

RD—Read Unprotected Data

The RD command reads a specified number of characters of unprotected data from a terminal, beginning at a specified location. For a roll screen device, RD is identical to RA.

RD Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22	23-24	25-26	27-30
WSC	*pc*	*sect*	*sc*	RD	*terminal*	*rr*	*cc*	*numb*

WSC
is the primary transaction identifier.

pc
is the primary cell identifier (the cell in which the command is to be executed).

sect
is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

RD
is the work session controller command.

terminal
is the terminal name used in this session.

rr
is the beginning row to be read. For roll screen terminals, specify 00.

cc
is the beginning column to be read. For roll screen terminals, specify 00.

numb
is the number of characters to be read. For static screen terminals, *numb* can be from 1 to 1920; for roll screen terminals, it can be from 1 to 102.

RD Acknowledgment Transaction Format (Required)

1-4	5-6	7-10	11-12	13-14	15-22	23-24	25-26	27-30	31-*n*
prit	*pc*	WSC	*sc*	RD	*terminal*	*rr*	*cc*	*numb*	*data*

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

RD
is the work session controller command.

terminal
is the terminal name used in this session.

rr
is the row where the cursor is positioned (00 for roll screen terminals).

cc
is the column where the cursor is positioned (00 for roll screen terminals).

numb
is the number of characters read.

data
is the data read.

RD Example

To read 5 unprotected characters from a terminal, 4978TERM, starting at row 5, column 10:

WSCbbbUSR1bbRD4978TERM05100005

The acknowledgment transaction is:

USR1bbWSCb??RD4978TERM12200005HELLO

The cursor was at row 12, column 20 when the 5 characters (HELLO) were read.

RS—Restore Data

The RS command restores data from the partitioned data set $.WSCIMG on disk or diskette. The data is taken from the member that has the same name as the terminal name used for this session. The data restored is whatever was saved with the last SV command. You can request that the data be transferred to a data area you specify or returned in the acknowledgment transaction. If you request that it be transferred to a data area, the command can be executed only in the cell in which it is issued.

RS Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22	23-24	25-26	27-28
WSC	*pc*	*sect*	*sc*	RS	*terminal*	*ad*	*ak*	*no*

WSC
is the primary transaction identifier.

pc
is the primary cell identifier (the cell in which the command is to be executed).

sect
is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

RS
is the work session controller command.

terminal
is the terminal name used in this session.

ad
is the address, in binary, where the data is to be stored. Specify 0 if the data is to be returned in the acknowledgment transaction.

ak
is the address key associated with this data address, in binary. Specify -1 (X'FFFF') if the data is to be returned in the acknowledgment transaction. (See the TCBGET instruction in *EDX Language Reference.*)

no
is the number, in binary, of 256-byte records to be restored, in the range 1-256. If data is returned in the acknowledgment transaction, the range is 1-8 records.

RS Acknowledgment Transaction Format (Required)

1-4	5-6	7-10	11-12	13-14	15-22	23-24	25-*n*
prit	*pc*	WSC	*sc*	RS	*terminal*	*rc*	*data*

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

RS
is the work session controller command.

terminal
is the terminal name used in this session.

rc
is a return code:

Y0—Operation was successful
E1—Incorrect transaction length
E2—Key specified is invalid
E3—Number of records specified is more than the maximum
E4—Data set member not defined
E5—Disk I/O error occurred
E7—Request is not for this cell

data
is the restored data. This field is optional.

RS Examples

1 To restore 3 records of data to location PROGDATA in partition 5:

PROGDATA DATA 384F'0'

WSCbbbPRM101RSTERM1bbb*xxyyzz*

where *xx* is the address of PROGDATA, *yy* is the binary value X'0004' (the address key is 1 less than the partition number), and *zz* is the binary value X'0003'.

The acknowledgment transaction is:

PRM101WSCb??RSTERM1bbbY0

Three records were read into location PROGDATA in partition 5 from member TERM1 of data set $.WSCIMG.

2 To restore 3 records of data and receive it in the acknowledgment transaction:

WSCbbbPRM101RSTERM1bbb*xxyyzz*

where *xx* is X'0000', *yy* is X'FFFF', and *zz* is X'0003'.

The acknowledgment transaction is:

PRM101WSCb??RSTERM1bbbY0*aa*

where *aa* is 768 bytes of restored data.

RT—Read Program Function Key Table

The RT command reads the transaction identifiers that are set for program function keys 1-5.

RT Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22	23-30
WSC	*pc*	*sect*	*sc*	RT	*terminal*	*image*

WSC
is the primary transaction identifier.

pc
is the primary cell identifier (the cell in which the command is to be executed).

sect
is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

RT
is the work session controller command.

terminal
is the terminal name used in this session.

image
is the name of the screen image.

RT Acknowledgment Transaction Format (Required)

1-4	5-6	7-10	11-12	13-14	15-22	23-26
prit	*pc*	WSC	*sc*	RT	*terminal*	*t1*

27-28	29-32	33-34	35-38	39-40	41-44	45-46	47-50	51-52
c1	*t2*	*c2*	*t3*	*c3*	*t4*	*c4*	*t5*	*c5*

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
: is the secondary transaction identifier.

sc
: is the secondary cell identifier (the cell where your original WSC transaction was executed).

RT
: is the work session controller command.

terminal
: is the terminal name used in this session.

t1
: is the transaction identifier used as a primary transaction identifier when PF1 is pressed after a WK command.

c1
: is the cell identifier of the cell to which transaction *t1* is sent.

t2
: is the transaction identifier used as a primary transaction identifier when PF2 is pressed after a WK command.

c2
: is the cell identifier of the cell to which transaction *t2* is sent.

t3
: is the transaction identifier used as a primary transaction identifier when PF3 is pressed after a WK command.

c3
: is the cell identifier of the cell to which transaction *t3* is sent.

t4
: is the transaction identifier used as a primary transaction identifier when PF4 is pressed after a WK command.

c4
: is the cell identifier of the cell to which transaction *t4* is sent.

t5
: is the transaction identifier used as a primary transaction identifier when PF5 is pressed after a WK command.

c5
: is the cell identifier of the cell to which transaction *t5* is sent.

RT Example

To read the table of transaction identifiers for PF1-5 for terminal SCRN01:

```
WSCbbbUSR2bbRTSCRN01bb
```

The acknowledgment transaction is:

USR2bbWSCb??RTSCRN01bbPGM1bbPGM201bbbbbbPGM401PGM5bb

The acknowledgment indicates the following assignment of transactions to PF keys:

PF1—transaction PGM1 in this cell.
PF2—transaction PGM2 in cell 01.
PF3—not set; uses the secondary transaction ID in the WK command.
PF4—transaction PGM4 in cell 01.
PF5—transaction PGM5 in this cell.

SC—Set Cursor

The SC command positions the cursor on a static screen terminal.

SC Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22	23-24	25-26
WSC	*pc*	*sect*	*sc*	SC	*terminal*	*rr*	*cc*

WSC
: is the primary transaction identifier.

pc
: is the primary cell identifier (the cell in which the command is to be executed).

sect
: is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
: is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

SC
: is the work session controller command.

terminal
: is the terminal name used in this session.

rr
: is the number of the row to which the cursor is to be positioned.

cc
: is the number of the column to which the cursor is to be positioned.

SC Acknowledgment Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
prit	*pc*	WSC	*sc*	SC	*terminal*

prit
: is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
: is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
 is the secondary transaction identifier.

sc
 is the secondary cell identifier (the cell where your original WSC transaction was executed).

SC
 is the work session controller command.

terminal
 is the terminal name used in this session.

SC Example

To set the cursor on terminal SCRN01 to row 0, column 0:

```
WSCbbbENTR01SCSCRN01bb0000
```

The acknowledgment transaction is:

```
ENTR01WSCb??SCSCRN01bb
```

indicating that the cursor was set.

SD—Start Device

The SD command starts an EDX terminal as a work session controller terminal. A station block with the terminal name to be used for this session is created.

SD Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22	23-30	31-33	34-41
WSC	*pc*	*sect*	*sc*	SD	*terminal*	*EDX name*	*wa*	*staname*

WSC
is the primary transaction identifier.

pc
is the primary cell identifier (the cell in which the command is to be executed).

sect
is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

SD
is the work session controller command.

terminal
is the terminal name to be used in this session.

EDX name
is the EDX terminal name. It cannot be the name of the Communications Facility system log device.

wa
is the size of the user work area to be appended to the station block when it is built, 1-128 bytes. This field is optional.

staname
is the station name (if any) by which this EDX terminal is defined to the Communications Facility as an emulated 3270 device. This field is optional and is valid only for 4978s, 4980s, 3101s, and printers.

SD Acknowledgment Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22	23
prit	*pc*	WSC	*sc*	SD	*terminal*	Y | N

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

SD
is the work session controller command.

terminal
is the terminal name used in this session.

Y | N
indicates whether or not the terminal was started.

SD Example

To start EDX terminal $SYSLOG with a terminal name of 4978TERM for this session, a 16-byte user work area, and an emulated 3270 device station name of T4978T1:

WSCbbbUSR1bbSD4978TERM$SYSLOGb016T4978T1b

The acknowledgment transaction is:

USR1bbWSCb??SD4978TERMY

The acknowledgment indicates that the start was successful.

SF—Set Forms

The SF command sets the forms parameters for a roll screen or output only terminal. Parameters for which you specify blanks retain their current values.

SF Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
WSC	*pc*	*sect*	*sc*	SF	*terminal*

23	24	25	26-28	29-31	32-34	35-37
l	*p*	*o*	*lin*	*tm*	*bm*	*rm*

WSC
: is the primary transaction identifier.

pc
: is the primary cell identifier (the cell in which the command is to be executed).

sect
: is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
: is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

SF
: is the work session controller command.

terminal
: is the terminal name used in this session.

l
: is the number of lines per inch—either 6 or 8. This field is ignored if the terminal is not a printer.

p
: is Y or N, indicating whether the terminal is to pause at the end of a page. This field is ignored if the terminal is a printer.

o
: is Y or N to indicate whether a line can overflow onto the next line.

lin
: is the number of lines per page.

tm
: is the top margin.

bm
is the bottom margin.

rm
is the right margin.

SF Acknowledgment Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
prit	*pc*	WSC	*sc*	SF	*terminal*

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

SF
is the work session controller command.

terminal
is the terminal name used in this session.

SF Example

To set the terminal 4974P1 to print at 8 lines per inch, with 35 lines per page and a right margin of 119:

WSCbbbPRNTLASF4974P1bb8bb035bbbbbb119

The acknowledgment transaction is:

PRNTLAWSCb??SF4974P1bb

indicating that the parameters have been modified.

SL—Set Station Name

The SL command notifies the work session controller that the terminal's emulated 3270 device station name has changed. This command is valid only for 4978s, 4980s, 3101s, and printers.

SL Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22	23-30
WSC	*pc*	*sect*	*sc*	SL	*terminal*	*staname*

WSC
: is the primary transaction identifier.

pc
: is the primary cell identifier (the cell in which the command is to be executed).

sect
: is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
: is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

SL
: is the work session controller command.

terminal
: is the terminal name used in this session.

staname
: is the station name by which this terminal is defined to the Communications Facility as an emulated 3270 device.

SL Acknowledgment Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
prit	*pc*	WSC	*sc*	SL	*terminal*

prit
: is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
: is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

SL
is the work session controller command.

terminal
is the terminal name used in this session.

SL Example

To notify the work session controller that the terminal for which the name 4978TERM is being used in this session has T4978T1 as its emulated 3270 device station name:

WSCbbbMEN1LHSL4978TERMT4978T1b

The acknowledgment transaction is:

MEN1LHWSCb??SL4978TERM

indicating that work session controller has been notified of the emulated 3270 station name.

SS—Start a Session

The SS command starts a session with a terminal. The work session controller executes an ENQT for the terminal and, optionally, sends it a screen image.

SS Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22	23	24-31
WSC	*pc*	*sect*	*sc*	SS	*terminal*	*w*	*priscreen*

WSC
is the primary transaction identifier.

pc
is the primary cell identifier (the cell in which the command is to be executed).

sect
is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

SS
is the work session controller command.

terminal
is the terminal name used in this session.

w
is Y or N, to indicate whether or not the work session controller is to wait until it can do the ENQT.

priscreen
is the name of a screen image in $.WSCIMG to be sent to the terminal. This field is optional and is valid only for static screen terminals.

SS Acknowledgment Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22	23
prit	*pc*	WSC	*sc*	SS	*terminal*	*s*

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

SS
is the work session controller command.

terminal
is the terminal name used in this session.

s
is Y or N, to indicate that the session is or is not started.

SS Example

To start a session with terminal 4978TERM, using image 4978PRI and waiting for the ENQT:

WSCbbbPROGbbSS4978TERMY4978PRIb

The acknowledgment transaction is:

PROGbbWSCb??SS4978TERMY

indicating that the session was successfully started.

ST—Set Program Function Key Table

The ST command sets the transaction identifiers of the transactions that are to be sent when PF keys 1-5 on a static screen terminal are pressed after a WK command. PF keys for which you use a blank transaction identifier will use the secondary transaction identifier in the WK command.

ST Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22	23-26
WSC	*pc*	sect	sc	ST	*terminal*	*t1*

27-28	29-32	33-34	35-38	39-40	41-44	45-46	47-50	51-52
c1	*t2*	*c2*	*t3*	*c3*	*t4*	*c4*	*t5*	*c5*

WSC
: is the primary transaction identifier.

pc
: is the primary cell identifier (the cell in which the command is to be executed).

sect
: is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
: is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

ST
: is the work session controller command.

terminal
: is the terminal name used in this session.

t1
: is the transaction identifier used as a primary transaction identifier when PF1 is pressed after a WK command.

c1
: is the cell identifier of the cell to which transaction *t1* is sent.

t2
: is the transaction identifier used as a primary transaction identifier when PF2 is pressed after a WK command.

c2
: is the cell identifier of the cell to which transaction *t2* is sent.

t3
: is the transaction identifier used as a primary transaction identifier when PF3 is pressed after a WK command.

c3
is the cell identifier of the cell to which transaction *t3* is sent.

t4
is the transaction identifier used as a primary transaction identifier when PF4 is pressed after a WK command.

c4
is the cell identifier of the cell to which transaction *t4* is sent.

t5
is the transaction identifier used as a primary transaction identifier when PF5 is pressed after a WK command.

c5
is the cell identifier of the cell to which transaction *t5* is sent.

ST Acknowledgment Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
prit	*pc*	WSC	*sc*	ST	*terminal*

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

ST
is the work session controller command.

terminal
is the terminal name used in this session.

ST Example

To set PF keys 1 and 3 to transaction identifier ACCT in this cell; set PF4 to transaction identifier MAIN in cell 0A; and leave PF keys 2 and 5 to use the secondary transaction identifier in the WK command:

```
WSCbbbPROGbbST4978TERMACCTbbbbbbbbACCTbbMAIN0Abbbbbb
```

The acknowledgment transaction is:

PROGbbWSCb??ST4978TERM

The PF key transaction identifiers have been set and will be used in subsequent WK commands.

SV—Save Data

The SV command saves data in the partitioned data set $.WSCIMG on disk or diskette. The data is written to the member that has the same name as the terminal name used for this session. You must already have allocated the member (using the $DIUTIL utility) before you enter the SV command. You can request that the data be transferred from a data area you specify or pass it in the SV transaction. If you request that it be transferred from a data area, the command can be executed only in the cell in which it is entered.

SV Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22	23-24	25-26	27-28	29-*n*
WSC	*pc*	*sect*	*sc*	SV	*terminal*	*ad*	*ak*	*no*	*data*

WSC
is the primary transaction identifier.

pc
is the primary cell identifier (the cell in which the command is to be executed).

sect
is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

SV
is the work session controller command.

terminal
is the terminal name used in this session.

ad
is the address of the data to be saved, in binary. Specify 0 if the data is in the transaction.

ak
is the address key associated with this data address, in binary. Specify -1 (X'FFFF') if the data is in the transaction. (See the TCBGET instruction in the *EDX Language Reference* manual.)

no
is the number, in binary, of 256-byte records to be saved, in the range 1-256. If data is passed in the transaction, the range is 1-8 records.

data
is the data to be saved. This field is optional.

SV Acknowledgment Transaction Format (Required)

1-4	5-6	7-10	11-12	13-14	15-22	23-24
prit	*pc*	WSC	*sc*	SV	*terminal*	*rc*

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

SV
is the work session controller command.

terminal
is the terminal name used in this session.

rc
is a return code:

Y0—Operation was successful
E1—Incorrect transaction length
E2—Key specified is invalid
E3—Number of records specified is more than 256
E4—Data set member not defined
E5—Disk I/O error occurred
E7—Request is not for this cell

SV Examples

1 To save 512 bytes of information from location SAVEDATA in partition 3:

SAVEDATA DATA 256 F'0'

WSCbAAPRM1ABSVTERM01bb*xxyyzz*

where *xx* is the address of SAVEDATA, *yy* is the binary value X'0002' (the address key is 1 less than the partition number), and *zz* is the binary value X'0002'.

The acknowledgment transaction is:

PRM1ABWSCb??SVTERM01bbE4

This acknowledgment indicates that the data was not saved, because the data set member TERM01 had not been allocated in $.WSCIMG.

2 To save 512 bytes of data passed in the transaction:

WSCbAAPRM1ABSVTERM01bb*xxyyzznn*

where *xx* is X'0000', *yy* is X'FFFF', *zz* is X'0002', and *nn* is 512 bytes of data.

The acknowledgment transaction is:

PRM1ABWSCb??SVTERM01bbE4bbY0

The data was saved in member TERM01 of data set $.WSCIMG.

TN—Sound Tone

The TN command sounds the tone on a terminal.

TN Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
WSC	*pc*	*sect*	*sc*	TN	*terminal*

WSC
is the primary transaction identifier.

pc
is the primary cell identifier (the cell in which the command is to be executed).

sect
is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

TN
is the work session controller command.

terminal
is the terminal name used in this session.

TN Acknowledgment Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
prit	*pc*	WSC	*sc*	TN	*terminal*

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

TN
is the work session controller command.

terminal
is the terminal name used in this session.

TN Example

To sound the tone on terminal 3101TERM:

WSCbbbbbbbbTN3101TERM

No acknowledgment transaction is received because the secondary transaction identifier is blank.

UK—Unlock Keyboard

The UK command unlocks the keyboard on a static screen terminal.

UK Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
WSC	*pc*	*sect*	*sc*	UK	*terminal*

WSC
is the primary transaction identifier.

pc
is the primary cell identifier (the cell in which the command is to be executed).

sect
is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

UK
is the work session controller command.

terminal
is the terminal name used in this session.

UK Acknowledgment Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
prit	*pc*	WSC	*sc*	UK	*terminal*

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

UK
: is the work session controller command.

terminal
: is the terminal name used in this session.

UK Example

To unlock the keyboard on terminal 4979T1:

```
WSCbbbUSERbbUK4979T1bb
```

The acknowledgment transaction is:

```
USERbbWSCb??UK4979T1bb
```

indicating that the keyboard has been unlocked.

US—End Lock Sequence

The US command ends a lock sequence for a static screen terminal. This instruction does not change the physical state of the terminal.

US Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
WSC	*pc*	*sect*	*sc*	US	*terminal*

WSC
is the primary transaction identifier.

pc
is the primary cell identifier (the cell in which the command is to be executed).

sect
is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

US
is the work session controller command.

terminal
is the terminal name used in this session.

US Acknowledgment Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
prit	*pc*	WSC	*sc*	US	*terminal*

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
  is the secondary cell identifier (the cell where your original WSC transaction was executed).

US
  is the work session controller command.

terminal
  is the terminal name used in this session.

US Example

To end a lock sequence for terminal 4979T1:

WSCbbbUSERbbUS4979T1bb

The acknowledgment transaction is:

USERbbWSCb??US4979T1bb

indicating that the lock sequence is ended.

WD—Write Unprotected Data

The WD command writes unprotected data to a terminal. If the terminal is a 3101 in block mode, the first character of the data you specify will be replaced by a field attribute character. To provide for output either to a 3101 in block mode or to a 4978 or 4980, specify a blank as the first character. When the data is written to a 3101, the first position will be an attribute character, which appears as a protected blank on the screen. When the data is written to a 4978 or 4980, the blank will simply be the first position of the unprotected data.

WD Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22	23-24	25-26	27-n
WSC	*pc*	*sect*	*sc*	WD	*terminal*	*rr*	*cc*	*data*

WSC
is the primary transaction identifier.

pc
is the primary cell identifier (the cell in which the command is to be executed).

sect
is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

WD
is the work session controller command.

terminal
is the terminal name used in this session.

rr
is the starting row. For roll screen terminals, specify 00.

cc
is the starting column. For roll screen terminals, specify 00.

data
is the data to be written.

WD Acknowledgment Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
prit	*pc*	WSC	*sc*	WD	*terminal*

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

WD
is the work session controller command.

terminal
is the terminal name used in this session.

WD Example

To write 'NAME ADDRESS' to a roll screen terminal:

```
WSCbbbbbbbbWDROLLSCRN0000NAMEbADDRESS
```

No acknowledgment is sent because the secondary transaction identifier is blank. The data was written starting at the next position on the screen.

WK—Wait for a Key

The WK command waits for a static screen terminal's PF key or ENTER to be pressed or for a timeout set by an IT command to occur. If the keyboard is locked, a WK command unlocks the keyboard until an interrupting key is pressed, and then locks it again.

WK Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
WSC	*pc*	*sect*	*sc*	WK	*terminal*

WSC
: is the primary transaction identifier.

pc
: is the primary cell identifier (the cell in which the command is to be executed).

sect
: is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
: is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

WK
: is the work session controller command.

terminal
: is the terminal name used in this session.

WK Acknowledgment Transaction Format (Required)

1-4	5-6	7-10	11-12	13-14	15-22	23-24	25-26	27-29
prit	*pc*	WSC	*sc*	WK	*terminal*	*rr*	*cc*	*ikc*

prit
: is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
: is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
: is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

WK
is the work session controller command.

terminal
is the terminal name used in this session.

rr
is the row where the cursor was when the key was pressed or the timeout occurred.

cc
is the column where the cursor was when the key was pressed or the timeout occurred.

ikc
is the interrupt key code: 0 for ENTER, 1-5 for PF keys, and 254 for a timeout.

WK Example

Assuming that, for terminal SCRN01:

- transaction identifier PGM2, cell 01 is set for PF2 in the program function key table, and
- a time interval of one minute has been set through an IT transaction,

then to wait for a PF or ENTER key to be pressed, or for the IT time interval to elapse:

WSCbbbMAINbbWKSCRN01bb

If PF2 is pressed before the time interval elapses, this transaction is sent to the program associated with transaction identifier PGM2 in cell 01:

PGM201WSCb??WKSCRN01bb0001002

indicating the cursor position was row 0, column 1.

If the time interval elapses, this transaction is sent to the program associated with transaction identifier MAIN in the cell where the WK command originated:

MAINbbWSCb??WKSCRN01bb2379254

indicating the cursor position was row 23, column 79.

If ENTER is pressed before the time interval elapses, this transaction is sent to the program associated with transaction identifier MAIN in the cell where the WK command originated:

MAINbbWSCb??WKSCRN01bb1000000

indicating the cursor position was row 10, column 0.

WP—Write Protected Data

The WP command writes protected data to a static screen terminal. If the terminal is a 3101 in block mode, the first character of the data you specify will be replaced by a field attribute character. To provide for output either to a 3101 in block mode or to a 4978 or 4980, specify a blank as the first character. When the data is written to a 3101, the first position will be an attribute character that appears as a protected blank on the screen. When the data is written to a 4978 or 4980, the blank will simply be the first position of the protected data.

WP Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22	23-24	25-26	27-*n*
WSC	*pc*	*sect*	*sc*	WP	*terminal*	*rr*	*cc*	*data*

WSC
is the primary transaction identifier.

pc
is the primary cell identifier (the cell in which the command is to be executed).

sect
is the secondary transaction identifier (used as the primary transaction identifier in the acknowledgment transaction).

sc
is the secondary cell identifier (the cell to which the acknowledgment is to be sent).

WP
is the work session controller command.

terminal
is the terminal name used in this session.

rr
is the starting row.

cc
is the starting column.

data
is the data to be written.

WP Acknowledgment Transaction Format

1-4	5-6	7-10	11-12	13-14	15-22
prit	*pc*	WSC	*sc*	WP	*terminal*

prit
is the primary transaction identifier (it was the secondary transaction identifier in the original WSC transaction you sent).

pc
is the primary cell identifier (it was the secondary cell identifier in the original WSC transaction you sent).

WSC
is the secondary transaction identifier.

sc
is the secondary cell identifier (the cell where your original WSC transaction was executed).

WP
is the work session controller command.

terminal
is the terminal name used in this session.

WP Example

To write 'COMPANY NAME' to terminal TERM1 starting at row 2, column 20 when you know that the terminal is not a 3101 in block mode:

```
WSCbbbSTRTSFWPTERM1bbb0220COMPANYbNAME
```

If the terminal may be a 3101 in block mode:

```
WSCbbbSTRTSFWPTERM1bbb0219bCOMPANYbNAME
```

In either case, the acknowledgment transaction is:

```
STRTSFWSCb??WPTERM1bbb
```

indicating that the data was written.

Appendix A. $.CFMENU Sample Program

The Communications Facility includes a sample application program, $.CFMENU, which communicates with users at Communications Facility terminals. The *Design and Installation Guide* explains how to run the sample program. The program is presented here to illustrate some of the programming techniques you may use in your application programs.

$.CFMENU Functions

$.CFMENU communicates with users at 3270 display stations and at 4978, 4980, 3101, and 7485 terminals being managed as if they were 3277s. It demonstrates the following application programming techniques:

- The use of a predefined panel to present a menu to application users. This panel is the one used as an example in the chapter "Creating and Managing 3270 Panels" on page 45.
- The use of the GET F instruction to retrieve user input to the panel.
- The use of the various PUT instructions to send additional data to the application users. In one case, the program modifies the predefined panel; in others, it builds a complete 3270 data stream that contains an error message.
- The use of the SEND M and RECEIVE M instructions to communicate with multiple application users.
- The use of the LOCATE instruction to find a station block in order to check or modify that station's attributes.
- The use of the SEND CP instruction to send commands to control the network configuration.
- Various error-handling and exception-handling techniques.

$.CFMENU Listing

The rest of this chapter shows the sample program listing.

```
         TITLE    '*** COMMUNICATIONS FACILITY SAMPLE PROGRAM'
MENU     PROGRAM  START,460,DS=(($.SYSPNL,##))
*
***************************************************************
* THIS IS AN APPLICATION PROGRAM THAT PROVIDES A HIGH LEVEL   *
* INTERFACE BETWEEN A TERMINAL OPERATOR AND THE               *
* COMMUNICATIONS FACILITY.  THE PROGRAM PRESENTS A MENU TO    *
* THE USER WITH THESE OPTIONS:                                *
*                                                             *
*  1. CONNECT TO A HOST FOR PASS-THROUGH TYPE OF OPERATIONS.  *
*                                                             *
*  2. CONNECT TO ANOTHER COMMUNICATIONS FACILITY APPLICATION  *
*     PROGRAM.                                                *
*                                                             *
*  3. LOAD THE FULL SCREEN TEXT EDITOR ($FSEDIT).             *
*                                                             *
*  4. DISCONNECT FROM THE COMMUNICATIONS FACILITY.            *
*                                                             *
* THE PROGRAM CAN BE USED FROM ANY COMMUNICATIONS FACILITY    *
* TERMINAL, EITHER A REAL 3276/3277 OR A 4978 OR 3101 BEING   *
* MANAGED AS IF IT WERE A 3277.                               *
*                                                             *
* THE PROGRAM USES THE FOLLOWING 3270 SCREEN FORMATTING       *
* FUNCTIONS OF THE COMMUNICATIONS FACILITY:                   *
*                                                             *
*  1. THE MENU IS A PANEL NAMED 'CFMENU' IN DATA SET          *
*     $.SYSPNL.  THE PANEL WAS CREATED BY MEANS OF THE        *
*     COMMUNICATIONS FACILITY PANEL DESIGN PROGRAM, $.PANEL.  *
*                                                             *
*  2. SUBROUTINE S$GETPNL IS USED TO FETCH THE PANEL FROM     *
*     THE DATA SET.                                           *
*                                                             *
*  3. PUT INSTRUCTIONS ARE USED TO SEND ADDITIONAL DATA TO    *
*     THE USER, AND GET FIELD IS USED TO RETRIEVE THE USER'S  *
*     INPUT.                                                  *
*                                                             *
* THE PROGRAM MUST BE DEFINED AS A COMMUNICATIONS FACILITY    *
* STATION WITH NAME = $.CFMENU AND TYPE = USER.               *
***************************************************************
*
***************************************************************
* COPY THE COMMUNICATIONS FACILITY SYSTEM EQUATES AND EDX     *
* SYSTEM EQUATES THAT ARE REQUIRED BY THE PROGRAM.            *
***************************************************************
*
         COPY   S$CFEQU       COMMUNICATIONS FACILITY EQUATES
         COPY   DSCBEQU       EDX DATA SET CONTROL BLOCK EQUATES
         COPY   TCBEQU        EDX TASK CONTROL BLOCK EQUATES
         COPY   CCBEQU        EDX TERMINAL CONTROL BLOCK EQUATES
*
***************************************************************
* COPY THE SUBROUTINE USED TO FETCH THE MENU PANEL.           *
***************************************************************
*
         COPY   S$GETPNL
         TITLE '*** COMMUNICATIONS FACILITY SAMPLE PROGRAM'
*
***************************************************************
* DATA DEFINITIONS.                                           *
***************************************************************
*
INBUFF   DEFINE BUFFER,SIZE=256  GENERAL I/O BUFFER
OUTBUFF  DEFINE BUFFER,SIZE=768  BUFFER USED FOR MENU PANEL
*
```

```
* FIRST PART OF PANEL IS THE FIELD TABLE.  EACH ENTRY CONTAINS
* THE FIELD'S SCREEN LOCATION, LENGTH, AND TYPE.  THIS PROGRAM
* REFERENCES ONLY THE SCREEN LOCATION (1ST TWO BYTES):
*
HOST      EQU    OUTBUFF          HOST CONNECT INPUT FIELD
APPL      EQU    HOST+4           PROGRAM CONNECT INPUT FIELD
EDIT      EQU    APPL+4           EDITOR WORKFILE INPUT FIELD
DISC      EQU    EDIT+4           DISCONNECT REQUEST INPUT FLD
ERRLOC    EQU    DISC+4           ERROR MESSAGE OUTPUT FIELD
*
PNLNAME   TEXT   'CFMENU          NAME OF PANEL
TXTFLD    TEXT   LENGTH=8         INPUT AREA FOR GET FIELD
RCODE     DATA   F'0'             RETURN CODE
SVCOUNT   DATA   F'0'             SAVE BUFFER COUNT FIELD
SVDATA@   DATA   F'0'             SAVE BUFFER DATA POINTER
WORK      DATA   F'0'             WORK AREA
FSPARM    DATA   F'0'             $FSEDIT PARAMETER
ORIGIN    TEXT   LENGTH=8         MESSAGE ORIGIN, STATION NAME
ORTYPE    DATA   X'0000'          ... AND TYPE/SUBTYPE
DEVSTAT   EQU    X'016C'          3270 DEVICE STATUS MESSAGE
DEVEND    EQU    X'C240'          DEVICE END STATUS/SENSE
*
*                                 STATION TYPE/SUBTYPES:
E3277     EQU    X'04E2'          EMULATED 3277 TERMINAL
PORT      EQU    X'C4D0'          EMULATED CHANNEL ATTACH PORT
SNA3277   EQU    X'12E2'          SNA LU - 3277 TERMINAL
SNA3278   EQU    X'12EA'          SNA LU - 3278 TERMINAL
SNA3279   EQU    X'12EB'          SNA LU - 3279 TERMINAL
D4978     EQU    X'0678'          4978 DEVICE
*
*                                 COMMAND PROCESSOR MESSAGES:
CPSTOP    TEXT   'P XXXXXXXX'     STOP COMMAND
STOPSTA   EQU    CPSTOP+2
CPSTART   TEXT   'S XXXXXXXX'     START COMMAND
STARTSTA  EQU    CPSTART+2
CPLINK    TEXT   'LINK XXXXXXXX XXXXXXXX BOTH'  LINK COMMAND
LINKSTA1  EQU    CPLINK+5
LINKSTA2  EQU    CPLINK+14
*
*                                 3270 CONTROL DATA:
          DATA   X'0201'          INSERT CURSOR ORDER
ICUR      DATA   X'1300'
*
          DATA   X'0201'          CLEAR KEY ATTENTION ID
CLEAR     DATA   X'6D02'
*
          DATA   X'0201'          BLANK FOR REPEAT-TO-ADDRESS
BLANK     DATA   X'403C'
*
*                                 USER MESSAGES
ERRMSG1   TEXT   'HOST TERMINAL NOT FOUND'
ERRMSG2   TEXT   'THAT IS NOT AN EMULATED TERMINAL'
ERRMSG3   TEXT   'PROGRAM NOT FOUND'
ERRMSG4   TEXT   'THAT IS NOT A PROGRAM'
ERRMSG5   TEXT   'CANNOT USE EDITOR AT 3277/3101'
ERRMSG6   TEXT   'LOAD OF $FSEDIT FAILED, RC =NNN'
MSG6RC    EQU    ERRMSG6+28
*
*                                 SYSTEM MESSAGES
SYSMSG1   TEXT   'ERROR ON LOCATE'
SYSMSG2   TEXT   'ERROR ON PANEL FETCH'
SYSMSG3   TEXT   'ERROR ON RECEIVE'
SYSMSG4   TEXT   'PROGRAM STOPPED'
*
*                                 TERMINAL I/O CONTROL BLOCK
DEVICE    IOCB
*
*
```

```
*************************************************************
* LOCATE THE PROGRAM'S STATION BLOCK; CREATE IT IF IT DOESN'T*
* EXIST.  THIS MAKES IT POSSIBLE TO START THE PROGRAM WITH   *
* EITHER THE CP START COMMAND, THE EDX $L COMMAND, OR $DEBUG.*
*************************************************************
*
START    EQU   *
         LOCATE  ST,#1,OPTION=CREATE,EXIT=ERROR1  FIND/CREATE
         IF    (MENU,EQ,-2)          STATION BLOCK CREATED
           MOVE  (Q$NAU,#1),X'018A'  SET NETWORK ADDRESS
         ENDIF
         IOR   (Q$STAT,#1),+Q#ACTIVE FLAG STATION ACTIVE
*
*************************************************************
* FETCH THE MENU PANEL FROM DATA SET $.SYSPNL INTO BUFFER    *
* 'OUTBUFF'.  THE FIRST PART OF THE PANEL IS A FIELD TABLE   *
* THAT CONTAINS THE SCREEN LOCATION OF THE FIELDS THAT ARE   *
* REFERENCED BY THE PROGRAM.  FOLLOWING THE FIELD TABLE IS   *
* THE 3270 DATA STREAM THAT PRODUCES THE MENU WHEN IT IS     *
* SENT TO THE USER'S TERMINAL.                               *
*************************************************************
*
         CALL  S$GETPNL,(OUTBUFF),(PNLNAME),(DS1),RCODE
         IF    (RCODE,NE,-1),GOTO,ERROR2    FETCH FAILED
*
*************************************************************
*    S T A R T   O F   M A I N   P R O G R A M   L O O P    *
*-----------------------------------------------------------*
* GET THE NEXT MESSAGE FROM THE PROGRAM'S QUEUE; IF THERE    *
* ARE NONE, WAIT FOR ONE.  SAVE THE STATION NAME OF THE      *
* MESSAGE ORIGINATOR SO THAT A RESPONSE CAN BE DIRECTED TO   *
* THE ORIGINATING TERMINAL.  EXIT TO 'ERROR3' IS TAKEN IF    *
* THE MESSAGE IS A STATUS MESSAGE -- A STOP OR HALT COMMAND. *
*************************************************************
*
INPUTM   EQU     *
         RECEIVE  M,INBUFF,ORIGIN=ORIGIN,WAIT=YES,EXIT=ERROR3
*
*************************************************************
* CHECK FOR A 3270 DEVICE STATUS MESSAGE, ONE IDENTIFIED BY  *
* X'016C' IN THE FIRST TWO BYTES.  THE DEVICE STATUS/SENSE   *
* INFORMATION IS IN BYTES 6 AND 7; X'C240' INDICATES DEVICE  *
* END, WHICH MEANS THAT THE DEVICE CHANGED FROM 'NOT READY'  *
* OR 'BUSY' TO 'READY'.  IF THE MESSAGE SHOWS DEVICE END,    *
* SEND THE MENU TO THE USER.  IGNORE ANY OTHER DEVICE STATUS *
* MESSAGE (THE I/O CONTROL PROGRAM HAS LOGGED IT).           *
*************************************************************
*
         IF    (INBUFF,EQ,+DEVSTAT)          DEVICE STATUS
           IF    (INBUFF+6,EQ,+DEVEND),GOTO,SENDMU  DEVICE END
           GOTO  INPUTM                      GET NEXT MESSAGE
         ENDIF
*
*************************************************************
* THE MESSAGE IS INPUT FROM THE USER, THE RESULT OF A 3270   *
* READ MODIFIED COMMAND.  IT CONTAINS AT LEAST 5 BYTES OF    *
* DATA:  STX, DEVICE ADDRESS, ATTENTION ID, AND ETX.  IF THE *
* USER ENDED THE READ WITH OTHER THAN THE CLEAR KEY OR A PA  *
* KEY, IT ALSO CONTAINS THE CURSOR ADDRESS.  IF IT CONTAINS  *
* NO MORE THAN THAT, THE USER DID NOT ENTER ANY DATA,        *
* PERHAPS BECAUSE HE HAS NOT YET RECEIVED THE MENU.          *
*************************************************************
*
         IF    (INBUFF+B$COUNT,LE,7),GOTO,SENDMU    SEND MENU
*
```

```
************************************************************
* DETERMINE WHICH FIELD OF THE MENU WAS MODIFIED BY THE USER *
* AND PROCEED ACCORDINGLY.  THE GET FIELD INSTRUCTION        *
* SEARCHES THE BUFFER FOR AN SBA (SET BUFFER ADDRESS) ORDER  *
* THAT MATCHES THE SCREEN LOCATION SPECIFIED BY THE 4TH      *
* OPERAND (THE SCREEN LOCATIONS ARE OBTAINED FROM THE FIELD  *
* TABLE OF THE MENU PANEL).  IF A MATCH IS FOUND, THE DATA   *
* FOLLOWING THE SBA IS MOVE TO THE TEXT AREA (TXTFLD), AND   *
* THE TEXT COUNT IS SET TO THE LENGTH OF THE DATA.  IF NO    *
* MATCH WAS FOUND, THE TEXT COUNT IS SET TO ZERO.            *
************************************************************
*
         MOVE  TXTFLD,X'40',(8,BYTE)          CLEAR INPUT FIELD
         GET   FIELD,TXTFLD,INBUFF,HOST*      HOST CONNECTION?
         IF    (TXTFLD-1,NE,0,BYTE),GOTO,HOSTCON
         GET   FIELD,TXTFLD,INBUFF,APPL*      CF APPLICATION?
         IF    (TXTFLD-1,NE,0,BYTE),GOTO,APPLCON
         GET   FIELD,TXTFLD,INBUFF,EDIT*      LOAD EDITOR?
         IF    (TXTFLD-1,NE,0,BYTE),GOTO,EDITCON
         GET   FIELD,TXTFLD,INBUFF,DISC*      DISCONNECT?
         IF    (TXTFLD-1,NE,0,BYTE),GOTO,DISCON
*
************************************************************
* SEND THE MENU TO THE USER.  S$GETPNL, WHICH WAS USED TO    *
* FETCH THE PANEL, SET THE BUFFER HEADER FIELDS TO ADDRESS   *
* THE 3270 DATA STREAM, SO THE BUFFER CAN BE USED AS THE     *
* THE SUBJECT OF THE SEND MESSAGE INSTRUCTION.               *
************************************************************
*
SENDMU   EQU   *
         SEND  MESSAGE,ORIGIN,OUTBUFF    SEND MENU TO USER
         GOTO  INPUTM                    GET NEXT MESSAGE
*
************************************************************
* THE USER WISHES TO CONNECT TO THE HOST.  INPUT (IN TXTFLD) *
* IS THE STATION NAME OF THE EMULATED TERMINAL TO LINK TO.   *
* LOCATE THAT STATION BLOCK AND CHECK THAT IT IS AN EMULATED *
* TERMINAL (AN EMULATED 3277, A PORT, OR AN SNA LU TERMINAL).*
************************************************************
*
HOSTCON  EQU     *
         LOCATE  ST,#1,TXTFLD            GET STATION BLOCK ADDR
         IF    (MENU,NE,-1)              STATION BLK NOT FOUND
           CALL  UERROR,(ERRMSG1),HOST   REPORT ERROR
           GOTO  INPUTM                  GET NEXT MESSAGE
         ENDIF
         IF    ((Q$TYPE,#1),NE,+E3277),AND,    NOT EMUL 3277
               ((Q$TYPE,#1),NE,+PORT),AND,     ... NOR PORT
               ((Q$TYPE,#1),NE,+SNA3277),AND,  ... NOR SNA 3277
               ((Q$TYPE,#1),NE,+SNA3278),AND,  ... NOR SNA 3278
               ((Q$TYPE,#1),NE,+SNA3279)       ... NOR SNA 3279
           CALL  UERROR,(ERRMSG2),HOST   REPORT ERROR
           GOTO  INPUTM                  GET NEXT MESSAGE
         ENDIF
*
************************************************************
* THE STATION IS AN EMULATED TERMINAL.  BUILD A LINK COMMAND *
* THAT LINKS IT AND THE USER'S TERMINAL TO EACH OTHER, AND   *
* SEND IT TO THE COMMAND PROCESSOR.  THIS METHOD OF LINKING  *
* CHANGES THE NETWORK CONFIGURATION BOTH IN STORAGE AND IN   *
* THE $.SYSNET DATA SET.                                     *
************************************************************
*
         MOVE  LINKSTA1,ORIGIN,(8,BYTE)  USER'S TERMINAL NAME
         MOVE  LINKSTA2,TXTFLD,(8,BYTE)  EMULATED TERMINAL NAME
         SEND  CP,,CPLINK,ACK=YES        SEND COMMAND, WAIT FOR
*                                        ... COMPLETION
*
```

```
*************************************************************
* THE LINK IS ESTABLISHED.  INITIATE COMMUNICATION BETWEEN   *
* THE TWO STATIONS BY SENDING A MESSAGE TO THE EMULATED      *
* TERMINAL THAT MAKES IT APPEAR THAT THE USER AT THE REAL    *
* TERMINAL PRESSED THE CLEAR KEY.                            *
*************************************************************
*
SENDAID  EQU    *
         PUT    AID,INBUFF,CLEAR          CLEAR KEY ATTENTION ID
         SEND   MESSAGE,TXTFLD,INBUFF,ORIGIN=ORIGIN
         GOTO   INPUTM                    GET NEXT MESSAGE
*
*************************************************************
* THE USER WISHES TO CONNECT TO AN APPLICATION PROGRAM.      *
* INPUT (IN TXTFLD) IS THE PROGRAM'S STATION NAME.           *
* LOCATE ITS STATION BLOCK AND CHECK THAT IT IS A PROGRAM    *
* (STATION TYPE = USER).                                     *
*************************************************************
*
APPLCON  EQU    *
         LOCATE  ST,#1,TXTFLD             GET STATION BLOCK ADDR
         IF     (MENU,NE,-1)              STATION BLK NOT FOUND
           CALL  UERROR,(ERRMSG3),APPL    REPORT ERROR
           GOTO  INPUTM                   GET NEXT MESSAGE
         ENDIF
         IF     ((Q$TYPE,#1),NE,+Q#USER,BYTE)  TYPE NOT USER
           CALL  UERROR,(ERRMSG4),APPL    REPORT ERROR
           GOTO  INPUTM                   GET NEXT MESSAGE
         ENDIF
*
*************************************************************
* THE STATION IS A PROGRAM.  LINK THE USER'S TERMINAL TO IT  *
* BY PUTTING THE NETWORK ADDRESS OF THE PROGRAM STATION IN   *
* THE TERMINAL STATION BLOCK.  THIS METHOD OF LINKING CHANGES*
* THE NETWORK CONFIGURATION IN STORAGE ONLY.  NOTE THAT      *
* THERE IS NO NEED TO LINK THE OTHER WAY (THE PROGRAM TO THE *
* TERMINAL) BECAUSE IT IS EXPECTED THAT AN APPLICATION       *
* PROGRAM USES THE ORIGIN OBTAINED ON A RECEIVE INSTRUCTION  *
* TO IDENTIFY THE STATION WITH WHICH IT IS COMMUNICATING.    *
*************************************************************
*
         LOCATE ST,#2,ORIGIN,EXIT=INPUTM  FIND USER'S STATION
         MOVE    (Q$DLV,#2),(Q$NAU,#1)      ESTABLISH LINK
         GOTO    SENDAID                    SIMULATE CLEAR KEY
*
*************************************************************
* THE USER WISHES TO CONNECT TO THE EDX TEXT EDITOR.         *
* INPUT (IN TXTFLD) IS THE NAME OF THE WORK FILE TO BE USED. *
* CHECK THAT THE USER'S TERMINAL IS A 4978; THE EDITOR CANNOT*
* BE USED WITH A 3277 OR A CHARACTER-MODE 3101.  THE TERMINAL*
* TYPE, ALONG WITH ITS STATION NAME, WAS OBTAINED FROM THE   *
* RECEIVE INSTRUCTION.                                       *
*************************************************************
*
EDITCON  EQU    *
         IF     (ORTYPE,NE,+D4978)        TERMINAL NOT A 4978
           CALL  UERROR,(ERRMSG5),EDIT    REPORT ERROR
           GOTO  INPUTM                   GET NEXT MESSAGE
         ENDIF
*
```

```
******************************************************************
* THE USER'S TERMINAL IS A 4978.  LOCATE ITS STATION BLOCK      *
* AND EXTRACT FROM IT THE EDX DEVICE NAME.  DISCONNECT THE      *
* TERMINAL FROM THE COMMUNICATIONS FACILITY BY SENDING A        *
* STOP COMMAND TO THE COMMAND PROCESSOR.  THEN ENQUEUE ON       *
* THE EDX TERMINAL AND ATTEMPT TO LOAD $FSEDIT.                 *
******************************************************************
*
         LOCATE ST,#2,ORIGIN,EXIT=INPUTM  GET STATION BLK ADDR
         MOVE   #2,($TCBCCB+Q$TCB,#2)     GET TCB ADDRESS
         MOVE   DEVICE,($CCBNAME,#2),(8,BYTE),FKEY=0
*                                         DEVICE NAME TO IOCB
         MOVE   STOPSTA,ORIGIN,(8,BYTE)   BUILD STOP COMMAND
         SEND   CP,,CPSTOP,ACK=YES        SEND COMMAND, WAIT
*                                         ... FOR COMPLETION
         ENQT   DEVICE                    ENQ ON EDX TERMINAL
         MOVE   LOAD+26,TXTFLD,(8,BYTE)   MOVE WORKFILE NAME
*                                         ... TO LOAD INSTR
*                                         ... AND LOAD EDITOR
LOAD     LOAD   $FSEDIT,FSPARM,DS=DUMMY,LOGMSG=NO,PART=ANY
         MOVE   RCODE,MENU                SAVE RETURN CODE
         IF     (RCODE,EQ,-1),GOTO,INPUTM LOADED OK, GET
*                                         ... NEXT MESSAGE
*
******************************************************************
* THE LOAD OF $FSEDIT FAILED.  RESTART THE USER'S TERMINAL      *
* AS A COMMUNICATIONS FACILITY TERMINAL.  THEN USE PUT FIELD    *
* INSTRUCTIONS TO MODIFY THE MENU PANEL (APPEND AN ERROR        *
* MESSAGE, APPEND THE NAME OF THE SPECIFIED WORK FILE, AND      *
* POSITION THE CURSOR AT THE WORK FILE NAME), AND SEND THE      *
* PANEL TO THE USER.                                            *
******************************************************************
*
         MOVE  STARTSTA,ORIGIN,(8,BYTE)   BUILD START COMMAND
         SEND  CP,,CPSTART,ACK=YES        SEND COMMAND, WAIT
*                                         ... FOR COMPLETION
         CONVTB MSG6RC,RCODE,FORMAT=(3,0,I)  LOAD RETURN CODE
         MOVE  SVCOUNT,OUTBUFF+B$COUNT    SAVE BUFFER COUNT
         MOVE  SVDATA@,OUTBUFF+B$DATA@    ... AND DATA POINTER
         PUT   FIELD,OUTBUFF,ERRMSG6,ERRLOC*  APPEND ERROR MSG
         PUT   FIELD,OUTBUFF,TXTFLD,EDIT* APPEND WORKFILE NAME
         PUT   FIELD,OUTBUFF,1CUR,EDIT*   SET CURSOR
         SEND  MESSAGE,ORIGIN,OUTBUFF     SEND MODIFIED MENU
*
******************************************************************
* RESTORE THE MENU TO ITS ORIGINAL FORM BY RESTORING THE        *
* COUNT AND DATA POINTER FIELDS OF THE BUFFER HEADER.  THIS     *
* EFFECTIVELY REMOVES THE APPENDED DATA.                        *
******************************************************************
*
         MOVE  OUTBUFF+B$COUNT,SVCOUNT    RESTORE BUFFER COUNT
         MOVE  OUTBUFF+B$DATA@,SVDATA@    ... AND DATA POINTER
         GOTO  INPUTM                     GET NEXT MESSAGE
*
******************************************************************
* THE USER WISHES TO DISCONNECT HIS TERMINAL FROM THE           *
* COMMUNICATIONS FACILITY.  DO SO BY SENDING A STOP COMMAND     *
* TO THE COMMAND PROCESSOR.                                     *
******************************************************************
*
DISCON   EQU    *
         MOVE   STOPSTA,ORIGIN,(8,BYTE)   BUILD STOP COMMAND
         SEND   CP,,CPSTOP                SEND COMMAND, DON'T
*                                         ... NEED TO WAIT
         GOTO   INPUTM                    GET NEXT MESSAGE
*
```

```
*****************************************************************
* UERROR: SUBROUTINE TO SEND AN ERROR MESSAGE TO THE USER.      *
*         PARAMETER-1 IS THE ADDRESS OF THE ERROR MESSAGE.      *
*         PARAMETER-2 IS THE SCREEN LOCATION FOR THE CURSOR.    *
*                                                               *
* PUT INSTRUCTIONS ARE USED TO BUILD THE 3270 DATA STREAM.      *
* THE 1ST ONE RESETS THE BUFFER HEADER FIELDS AND MOVES A       *
*   3270 WRITE COMMAND AND WRITE CONTROL CHAR TO THE BUFFER.    *
* THE 2ND ONE PUTS THE ERROR MESSAGE IN THE BUFFER WITH AN      *
*   SBA ORDER FOR THE SCREEN LOCATION AT WHICH THE MESSAGE      *
*   IS TO APPEAR.  THIS LOCATION IS AS DEFINED IN THE MENU      *
*   PANEL FIELD TABLE.                                          *
* THE 3RD ONE PUTS A 'REPEAT TO ADDRESS' ORDER IN THE BUFFER    *
*   THAT CAUSES THE SCREEN TO BE CLEARED FROM THE END OF THE    *
*   MESSAGE TO THE END OF THE SCREEN.  THIS IS DONE IN CASE     *
*   A PRIOR, LONGER ERROR MESSAGE WAS SENT TO THE USER.         *
* THE 4TH ONE PUTS AN 'INSERT CURSOR' ORDER IN THE BUFFER       *
*   WITH AN SBA ORDER FOR THE SCREEN LOCATION SPECIFIED BY      *
*   THE CALLER.                                                 *
*****************************************************************
*
         SUBROUT  UERROR,MSG@,CLOC
         PUT    COMMAND,INBUFF,OPTION=(TONE)  WRITE AND SOUND
*                                             ... ALARM
         PUT    FIELD,INBUFF,MSG@*,ERRLOC*    ERROR MESSAGE
         PUT    REPEAT,INBUFF,BLANK,ROW=1,COLM=1 CLEAR ERR FLD
         PUT    FIELD,INBUFF,ICUR,CLOC*       INSERT CURSOR
         SEND   MESSAGE,ORIGIN,INBUFF         SEND TO USER
         RETURN
*
*****************************************************************
* LOG SYSTEM ERRORS, USING THE XCODE FIELD TO REPORT THE        *
* RETURN CODE THAT FURTHER IDENTIFIES THE CAUSE OF THE ERROR.*
*****************************************************************
*
ERROR1   EQU    *           ERROR ON LOCATE OF PROGRAM'S STATION
         SEND   ERROR,1,SYSMSG1,ID=C'UM',XCODE=MENU*  LOG ERROR
         GOTO   EXIT                                  STOP
*
ERROR2   EQU    *           ERROR ON FETCH OF PANEL 'CFMENU'
         SEND   ERROR,2,SYSMSG2,ID=C'UM',XCODE=RCODE* LOG ERROR
         GOTO   STOP                                  STOP
*
ERROR3   EQU    *           ERROR/EXCEPTION ON RECEIVE MESSAGE
         IF     (MENU,EQ,6)           RECEIVED A STATUS MESSAGE
           IF     (INBUFF,EQ,C'H '),  IF IT'S A HALT COMMAND
                OR,(INBUFF,EQ,C'P '), ... OR A STOP COMMAND,
                GOTO,STOP             ... STOP THE PROGRAM
         ENDIF
         SEND   ERROR,3,SYSMSG3,ID=C'UM',XCODE=MENU*  ELSE LOG
         GOTO   INPUTM                                CONTINUE
*
*****************************************************************
* THE PROGRAM HAS EITHER BEEN TOLD TO STOP OR HAS DECIDED TO *
* STOP BECAUSE OF A SYSTEM ERROR.  USE THE LOCATE INSTRUCTION*
* WITH THE PURGE OPTION TO PURGE PENDING MESSAGES FROM ITS    *
* QUEUE AND TO DELETE ITS STATION BLOCK.                        *
*****************************************************************
*
STOP     EQU       *
         LOCATE    ST,#1,OPTION=PURGE  PURGE MSGS & STATION BLK
*
EXIT     EQU       *
         SEND      LOG,4,SYSMSG4,ID=C'UM'  LOG TERMINATION
                   PROGSTOP LOGMSG=NO
                   ENDPROG
                   END
```

Glossary

This is a glossary of technical Communications Facility terms that appear in the book. Only terms unique to the Communications Facility are defined here. For definitions of Event Driven Executive terms, consult the *EDX System Guide*; for definitions of 3270 terms, see the *3270 Component Description* manual.

$.CF. The name given to the control program after it is loaded under the name $.CFD or $.CFS.

$.CFD. The version of the control program for Communications Facility systems that use disk queuing of messages.

$.CFMENU. A sample application program, distributed as part of the Communications Facility, that demonstrates how to communicate with users at 3270-type terminals.

$.CFS. The version of the control program for Communications Facility systems using storage queueing of messages.

$.CONFIG. A utility program that allows the user to define and modify stations and maintain the system message data set.

$.DISP. The Communication Facility system station.

$.IO0AB0. The input/output control program that manages communication between Series/1s attached to a Local Communications Controller.

$.IO0AB8. The input/output control program that manages X.25 packet level communication between a Series/1 and a DTE connected by an HDLC line, with or without an intervening X.25 packet-switching data network.

$.IO14E8. The input/output control program that provides 3270 emulation when a Series/1 is connected to a host processor that uses SNA.

$.PANEL. An interactive program for creating panels to be displayed at a 3270-type terminal.

$.PD. The program dispatcher; the program that manages the processing of transactions.

$.PNLUT1. The utility program that prints descriptions of panels created by the $.PANEL program.

$.SYSMSG. The data set that contains the text of error messages and informational messages issued by the Communications Facility and, optionally, by user programs.

$.SYSNET. The data set that contains the definitions of all the stations in a node and of remote stations that will be communicated with from that node.

$.SYSPD. A data set containing CP commands, path definitions, transaction definitions, and transactions that are to be processed when the program dispatcher is started.

$.SYSPNL. The data set that contains panels displayed by $.PANEL, $.CFMENU, and $.IO014E8 programs.

$.SYSX25. The data set that contains two-digit call IDs and their associated X.25 network addresses used during call establishment for switched virtual circuits. As shipped, it contains 10 records of /*.

$.UT1. A utility program that allows access to various Communications Facility functions for diagnostic purposes.

$.UT2. The utility program that allows the user to examine and purge messages on the $.WASTE queue or any other disk queue.

$.WASTE. The station to which undeliverable messages are sent.

$.WSC. The work session controller; the part of the Communications Facility that allows an application program to communicate with multiple EDX devices attached to any Series/1 in the network.

$.WSCIMG. The data set that contains images that can be displayed through the work session controller and members used to save data for transaction-processing programs.

$.WSCUT1. The utility program that allows a user to convert a screen image, prepared through the EDX $IMAGE utility, for use by the work session controller.

$.WSMENU. The program that can be used to start a session between an EDX terminal and work session controller application programs.

#L. A symbolic address used in language extension instructions to refer to the station block associated with the current task.

#T. A symbolic address used in language extension instructions to refer to the current task control block.

ACTIVATE T. The instruction that activates and deactivates tasks.

alternate link vector The network address of a station that is the alternate destination for messages sent to a particular station. Undeliverable transactions and X.25 control messages are sent to a station's alternate destination.

BCUG (bilateral closed user group). The X.25 facility that allows two DTEs to establish communication with each other by simply specifying a BCUG ID. Additional benefits vary with the network provider.

BI. The work session controller command that fetches a screen image form $.WSCIMG and displays it on the terminal.

bilateral closed user group (BCUG). The X.25 facility that allows two DTEs to establish communication with each other by simply specifying a BCUG ID. Additional benefits vary with the network provider.

BRB (buffer reference block). The first 4 words of a workspace pool, used to control the allocation of buffers and workspaces from the pool.

buffer, Communications Facility. A storage area, from 1 to 32K bytes long, preceded by a buffer header.

buffer header. Five words preceding a Communications Facility buffer that contain information about the size and content of the buffer.

buffer reference block (BRB). The first 4 words of a workspace pool, used to control the allocation of buffers and workspaces from the pool.

call accept control message. The Communications Facility message sent by an application program to indicate it is ready to receive data from a remote DTE through a switched virtual circuit. Call accept is sent in response to an incoming call control message.

call connected control message. The Communications Facility message sent by the X.25 I/O control program to a STD+ circuit station's direct or alternate link to indicate that the remote DTE has accepted the call initiated by the circuit station. The circuit station may now send data through the network.

call ID. A user-defined two-digit number that represents an X.25 network address. Call IDs are used when defining a switched virtual circuit station if calls are to be sent to or received from a specific X.25 network address. Call IDs are associated with X.25 addresses in the $.SYSX25 data set.

call request control message. The Communications Facility message sent by an application program to a circuit station with a contact type of USERINIT to initiate a virtual call to a remote DTE.

cause code. A 1-byte code in a restart, clear, or reset packet that indicates the reason for the restart, clear, or reset. The X.25 I/O control program includes this code in log and control messages that it sends as a result of receiving them from XHCS.

CC. The work session controller command that controls the carriage of a roll screen or hard copy terminal.

CCITT. International Telephone and Telegraph Consultative Committee. An organization of common carriers and PTTs whose main goal is to recommend standards that facilitate interconnection of communications equipment.

CD. The work session controller command that sets the unprotected data areas on a terminal to nulls.

cell. A node in the Communications Facility configuration in which the program dispatcher runs or a non-Series/1 host system where transactions are processed.

cell identifier. A 2-character name that uniquely identifies a cell.

CFBUF. The message buffer pool; a workspace pool in the Communications Facility control program that contains storage-queued messages.

circuit station. A station that represents an X.25 virtual circuit. See also *virtual circuit.*

clear confirmation control message. The Communications Facility message sent by the X.25 I/O control program to the STD+ circuit station's direct or alternate link informing it that the switched virtual circuit has been cleared as a result of a clear request control message sent by the circuit station.

clear indication control message. The Communications Facility message sent by the X.25 I/O control program to a STD+ circuit station informing it that the remote DTE has issued a clear request for the switched virtual circuit.

clear request control message. The Communications Facility message sent by an application program to a STD+ circuit station as a negative response to an incoming call or at any time it wants to clear the switched virtual circuit.

closed user group (CUG). The X.25 facility that defines a group of DTEs that can communicate with each other. The number of DTEs that can be in a group and additional benefits vary with the network provider. Contrast with *bilateral closed user group.*

command message. A message, the content of which is a CP command.

command processor. A part of the Communications Facility that processes CP and, with the program dispatcher, PD commands.

command processor (CP) commands. A set of Communications Facility commands used to define and control the Communications Facility configuration and display information about it.

command-processing program. A program that processes a particular CP command. The command processor controls loading and execution of command-processing programs as it receives commands.

Communications Facility buffer. A storage area, from 1 to 32K bytes long, preceded by a buffer header.

Communications Facility terminal. A device defined to the Communications Facility, controlled by an I/O control program, and accessed from a program through SEND and RECEIVE instructions.

configuration processor ($.CONFIG). A utility program used to define and modify stations and maintain the system message data set.

control message. A message defined by the Communications Facility that contains information related to controlling the X.25 network. Circuit stations with a usage type of STD+ and the X.25 I/O control program can send and receive these messages using the SEND S and SEND SM instructions. The length of the control message varies depending on which type it is. See also the name of the individual control message.

CP commands. A set of Communications Facility commands used to define and control the Communications Facility configuration and display information about it.

CUG (closed user group). The X.25 facility that defines a group of DTEs that can communicate with each other. The number of DTEs that can be in a group and additional benefits vary with the network provider. Contrast with *bilateral closed user group.*

data circuit-terminating equipment (DCE). The equipment installed at the user's premises that provides all the functions required to establish, maintain, and terminate a connection, including the signal conversion and coding between the data terminal equipment (DTE) and the line. In the Communications Facility, software functions provide a connection point for devices capable of interfacing to a packet-switching data network as DTEs. The Communications Facility does not provide all the DCE support defined by Recommendation X.25; it cannot be an X.25 network.

data message. A message, the content of which is user data to be sent from one station to another.

data terminal equipment (DTE). That part of a data station that serves as a data source, data sink, or both, and provides for the data communication control function according to protocols. In the Communications Facility, DTE is hardware or software that is capable of attaching to an X.25

DDM (device descriptor module). An XHCS module that defines an HDLC line. The name of the Communications Facility line station representing the line must have the same name as its DDM.

DEFINE BUFFER. The instruction that defines a Communication Facility buffer within a program.

DEFINE BUFFERPOOL. The instruction that defines a workspace pool within a program. This is the same instruction as DEFINE POOL and DEFINE WORK.

DEFINE POOL. The instruction that defines a workspace pool within a program. This is the same instruction as DEFINE BUFFERPOOL and DEFINE WORK.

DEFINE Q. The instruction that defines a queue control block.

DEFINE WORK. The instruction that defines a workspace pool within a program. This is the same instruction as DEFINE BUFFERPOOL and DEFINE POOL.

delivery confirmation bit (D-bit). A bit in an X.25 packet header that instructs the network to wait until delivery to the remote data terminal equipment has been confirmed before confirming delivery to the sending data terminal equipment. As data circuit-terminating equipment (DCE), the Communications Facility confirms only that the SEND M instruction to the local application program has completed successfully; it does not wait for confirmation of receipt of the message.

design indicator. One of four symbols used during $.PANEL execution to delimit fields and define their type.

design indicators phase. The phase of $.PANEL execution during which the user specifies which characters will be used as design indicators.

device descriptor module (DDM). An XHCS module that defines an HDLC line. The name of the Communications Facility line station representing the line must have the same name as its DDM.

device station. A station that represents a Series/1 terminal or printer to be managed as if it were a 3270 device in the Communications Facility configuration.

device type. The combination of station type and station subtype for a Series/1 terminal or printer being managed as if it were a 3270 device. Device type indicates which input/output control program is to control a particular device.

diagnostic aid utility ($.UT1). A utility program that allows access to various Communications Facility functions for diagnostic purposes.

diagnostic code. An optional 1-byte code in restart, reset, and clear packets that gives information about the reason for the restart, reset, or clear. The X.25 I/O control program includes this code, if XHCS sent it, in the log and control messages.

direct link vector. The network address of a station that is the default destination for messages sent by a particular station.

disk queue. A message queue on disk, used to hold low-priority messages destined for a particular station until the station is ready to receive them.

disk-queue data set. A data set used to hold low-priority messages destined for a particular station until the station is ready to receive them.

disk-queue data set initialization utility ($.DSINIT). The utility program that initializes a data set for the disk queuing of messages.

disk-queue file control block. A control block that contains information about a station that has a disk queue.

dispatcher, message. The part of the Communications Facility that determines the final destination of a message and routes it through the system to that destination.

dispatcher, program ($.PD). The part of the Communications Facility that manages the processing of transactions.

DTE (data terminal equipment). That part of a data station that serves as a data source, data sink, or both, and provides for the data communication control function according to protocols. In the Communications Facility, DTE is hardware or software that is capable of attaching to an X.25

dynamic program dispatcher. See *program dispatcher.*

EDL language extensions. A set of Event Driven Executive Language (EDL) instructions that perform various Communications Facility functions.

EDX terminal. A terminal defined to the EDX operating system and used to perform EDX system functions.

emulation, 3270. The facility that allows a host processor to communicate with a Series/1 as if it were communicating with a 3270 system; also, the I/O control program that provides 3270 emulation over a BSC line ($.IO0AE0).

error indication control message. The Communications Facility message sent by the X.25 I/O control program to an STD+ circuit station's direct or alternate link to indicate that the I/O control program rejected a control message or that it could not initiate a call for an SVC circuit station.

error message. A message sent to the Communications Facility log device to indicate that an error has occurred.

ES. The work session controller command that ends a session with a terminal.

facilities. A set of optional characteristics and capabilities available from the network provider to switched virtual circuits during call establishment. In the Communications Facility, these facilities may be included in the SVC circuit station definition and/or in a call request control message from an application program. See also the individual facility.

fast select (FS). The X.25 facility that allows data to be appended to a call request, call clear, or call accept packet. Contrast with *fast select restricted.*

fast select restricted (FSR). The X.25 facility that allows data to be appended to a call request or call clear packet. Call accept packets are not allowed. Contrast with *fast select.*

field attributes phase. The phase of $.PANEL execution during which the user specifies the attributes of the fields that make up the panel being created.

field fill indicator. A character used during $.PANEL execution to indicate trailing field positions that are to appear as blanks in the panel being created.

field table. A table of information about the fields in a panel.

FREE B. The instruction that returns a buffer to the workspace pool from which it was acquired.

FREE S. The instruction that returns storage to the workspace pool from which it was acquired. This instruction is the same as FREE W.

FREE W. The instruction that returns storage to the workspace pool from which it was acquired. This instruction is the same as FREE S.

FS (fast select). The X.25 facility that allows data to be appended to a call request, call clear, or call accept packet. Contrast with *fast select restricted.*

FSR (fast select restricted). The X.25 facility that allows data to be appended to a call request or call clear packet. Call accept packets are not allowed. Contrast with *fast select.*

FT. The work session controller command that reads the field table associated with a screen image in $.WSCIMG.

GET A. The instruction that gets the address of a system control block or table.

GET AI. The instruction that retrieves the attention ID (AID) byte from a 3270 data stream.

GET B. The instruction that gets a buffer from a workspace pool.

GET F. The instruction that moves a data field from a Communications Facility buffer to an EDX text area.

GET Q. The instruction that gets an element from a queue.

GET S. The instruction that gets storage from the system storage pool, S$POOL.

GET W. The instruction that gets storage from a workspace pool.

GOTEST. The PD command that allows execution of a program whose transaction identifier is in test mode.

H. (1) The CP command that removes a station from the active network. (2) The PD command that lists all the PD commmands and their functions.

halted station. A station whose station block has been deleted form S$POOL.

HDLC (high-level data link control). The group of standards defining the link level for communications with a public data network. As defined by Recommendation X.25, LAPB conforms to one subset of standards and SDLC normal response mode conforms to a different subset. The Communications Facility adheres only to the LAPB subset.

header, buffer. Five words preceding a Communications Facility buffer that contain information about the size and content of the buffer.

header, message. 24 bytes at the beginning of a message that contain such information as its origin, destination, and priority.

HELP. The CP command that lists all the CP commands and their functions.

high-speed loader ($.HSL). A small, efficient loader used by the program dispatcher to load transaction-processing programs.

host processor. A computer in a Communications Facility configuration where control functions are performed; it may be a Series/1 or another type of computer.

image library. A library of screen images in data set $.WSCIMG that can be displayed through the work session controller BI command.

image library management utility ($.WSCUT1). A utility program that converts a screen image that was created by the EDX $IMAGE program and stores it in $.WSCIMG for use by the work session controller. It is also used to display the image.

incoming call control message. The Communications Facility message sent by the X.25 I/O control program to a STD+ circuit station to indicate that a remote DTE wishes to begin communications.

input hold. A condition of a station in which all messages it sends are discarded. Messages sent by a station on input hold are discarded.

input message sequence number. The number of messages sent by a station.

input/output control program (IOCP). A program that handles transmission of messages to and from a particular type of device or line in a Communications Facility configuration.

interrupt confirmation control message. The Communications Facility message sent by the X.25 I/O control program to a STD+ circuit station informing it that the remote data terminal equipment received an interrupt control message sent by the circuit station.

interrupt control message. The Communications Facility message sent by an application program to a STD+ circuit station or by the X.25 I/O control program to a STD+ circuit station's direct or alternate link. The interrupt control message includes 1 byte of data that is sent across the X.25 circuit without flow control.

IOCP (input/output control program). A program that handles transmission of messages to and from a particular type of device or line in a Communications Facility configuration.

IT. The work session controller command that sets a timer for a static screen terminal.

language extensions. A set of Event Driven Executive Language (EDL) instructions that perform various Communications Facility functions.

LAPB (link access procedure balanced). The link protocol used by the XHCS for controlling the X.25 network access link.

LI. The work session controller command that links to another program.

line station. A station that represents a telecommunication line in a Communications Facility configuration.

line type. The combination of station type and station subtype for a communication line. Line type indicates which I/O control program (such as point-to-point or 3270 control) is to control a particular line.

linked station. A station that has a single specified station as the default destination of messages it sends.

LK. The work session controller command that locks the keyboard on a static screen terminal.

local node. The node from which the Communications Facility configuration is being viewed.

local station. A station that exists at the local node.

LOCATE LU. See *LOCATE ST.*

LOCATE NA. The instruction that locates a station block, given the network address of the station.

LOCATE QN. See *LOCATE ST.* The instruction is the same as LOCATE LU and LOCATE ST.

LOCATE ST. The instruction that locates a station block, given the name of the station; may also be used to create and delete station blocks. The instruction is the same as LOCATE LU and LOCATE QN.

log message. A message that is sent to the Communications Facility system log.

log processor. The part of the Communications Facility that formats error and informational messages and sends them to the system log.

logical unit, Communications Facility. See *station.*

logical unit (LU) station. An SNA logical unit (a terminal or a printer) in a Communications Facility configuration. Stations were called logical units in previous Communications Facility field developed programs.

LS. The work session controller command that sets a lock sequence for a static screen terminal.

LU (logical unit) station. A station that represents an SNA logical unit (a terminal or a printer) in a Communications Facility configuration.

mapped partition. A partition that a contains the common area (system tables and station blocks).

MENU. (1) The transaction identifier associated with the Communications Facility program $.WSMENU. (2) The work session controller high-level language subroutine that enables the application program to end its own operation and return control to the primary application load program.

message. A unit of data to be transmitted from one station to another.

message buffer pool (CFBUF). A workspace pool in the Communications Facility control program that contains storage-queued messages.

message data set ($.SYSMSG). The data set that contains the text of error messages and informational messages issued by the Communications Facility and, optionally, by user programs.

message dispatcher. The part of the Communications Facility that determines the final destination of a message and routes it through the system to the destination.

message header. 24 bytes at the beginning of a message that contain such information as the origin, destination, and priority of the message.

message priority. An attribute of a message that determines where it is placed in the destination station's message queue.

message queue. A queue of messages destined for a single station, either in processor storage or on disk.

message queue management utility ($.UT2). A utility program that allows the user to examine and purge messages on the $.WASTE queue or any other disk queue.

message sequence number. A number associated with a message representing its sequence with respect to its origin.

message station. A queue of messages, not associated with a Communications Facility program or device.

message type. An attribute of a message that indicates whether it is a data, command, log, transaction, or status message.

MOV. The instruction that moves data, allowing indirect and indexed moves.

name, station. A 1- to 8-character alphameric value that uniquely identifies each station in a node.

NAU. See *network address.*

network address. A 4-character hexadecimal value that uniquely identifies a station in the network. The first two characters are the node assignment, and the last two are the station address.

network configuration data set ($.SYSNET). The data set that contains the definitions of all the stations in a node and of remote stations that will be communicated with from that node.

node. A Series/1 in the Communications Facility configuration.

node station. A station that represents a remote node Communications Facility configuration.

nontransparent mode. A mode of BSC transmission that prohibits bit patterns with a value less than X'40' from being transmitted as data.

output hold. A condition of a station in which messages can be sent to it, but it can't receive messages from its queue.

output options phase. The phase of $.PANEL execution during which the user specifies actions (such as sounding the alarm or unlocking the keyboard) to be taken when the panel is displayed.

P. (1) The CP command that stops a station. (2) The PD command that deactivates a path or transaction.

packet. The basic transmission unit on a data link accessing an X.25 network. See also *packet size.*

packet size. The size of the largest data packet sent to an X.25 network. The packet size is defined in the circuit or DxE line station definitions.

packet switching. The process of routing and transferring data by means of addressed packets so that a channel is occupied only during the transmission of a packet.

packet-switching data network (PSDN). A communications network that uses the mechanism of packet switching to transmit data. See also *packet switching.*

panel. A screen image for a 3270 display station or a Series/1 device being managed as a 3270 display station.

panel data set ($.SYSPNL). The data set that contains panels displayed by the $.PANEL, $.CFMENU and $.IO014E8 programs.

panel design aid ($.PANEL). An interactive program for creating panels to be displayed at a 3270-type terminal.

panel layout phase. The phase of $.PANEL execution during which the user specifies the content of the panel being created.

panel print utility ($.PNLUT1). The utility program that prints descriptions of panels created by the $.PANEL program.

panel specification phase. The phase of $.PANEL execution in which the user identifies the panel to be designed.

passthrough control message. The Communications Facility message that contains user-defined control information. The X.25 I/O control program sends this message to the circuit station's direct or alternate link when it receives a message with the Q-bit on in an incoming data packet. The application program sends this message to a STD+ circuit when it wants the data sent in a data packet with the Q-bit on.

path. The route used to send a transaction from the program dispatcher in one cell to the program dispatcher in another cell or to a host transaction-processing system.

PATH. A statement in $.SYSPD that defines a path.

path definition table. A table of the paths the program dispatcher in a cell uses to route transactions to other cells.

path table. Synonymous with *path definition table.*

PD. The work session controller command that stops a work session controller terminal and returns it to EDX control.

PD commands. A subset of Communications Facility CP commands used to control the operation of the program dispatcher.

permanent virtual circuit (PVC). A permanent virtual connection that provides services similar to a leased line. Data sent to the network through a logical channel being used as a PVC is always delivered to a specific logical channel at a specific DTE destination in the network. In the Communications Facility, PVC circuit stations represent permanent virtual circuits.

physical unit (PU) station. A station that represents an SNA physical unit in a Communications Facility configuration.

preferred path. The path used to route transactions for unknown cells.

primary cell identifier. The field of a transaction that contains the identifier of the cell in which the transaction is to be processed.

primary transaction identifier. The field of a transaction that contains the transaction identifier.

priority, message. An attribute of a message that determines where it is placed in the destination station's message queue.

program dispatcher ($.PD). The part of the Communications Facility that manages the processing to transactions.

program dispatcher (PD) commands. A subset of Communications Facility CP commands used to control the operation of the program dispatcher.

program dispatcher data set ($.SYSPD). The data set containing CP commands, path definitions, transaction definitions, and transactions, that are to be processed when the program dispatcher is started.

program station. See *user station.*

prompt screen, SNA. An optional message put out by SNA logical units to prompt the terminal operator to log on to a host SNA application.

protected field. A field on a display screen in which the user is not allowed to enter data; also, the definition of such a field in a panel or an EDX screen image.

protocol identifier (ID). The optional 4-byte field included in the user data portion of a call request or incoming call packet that provides an additional means of screening incoming calls from an X.25 network. In the Communications Facility, switched virtual circuit station definitions may include a protocol ID.

PU (physical unit) station. A station that represents an SNA physical unit in a Communications Facility configuration.

PUT AID. The instruction that moves a 3270 attention ID (AID) byte into a Communications Facility buffer.

PUT CO. The instruction that moves a 3270 write command into a Communications Facility buffer.

PUT CURS. The instruction that moves a 3270 insert cursor order into a Communications Facility buffer.

PUT DLEETB. The instruction that moves a DLE and an ETB into a Communications Facility buffer.

PUT DLEETX. The instruction that moves a DLE and an ETX into a Communications Facility buffer.

PUT DLESTX. The instruction that moves a DLE and an STX into a Communications Facility buffer.

PUT ERA. The instruction that moves a 3270 erase unprotected to address order into a Communications Facility buffer.

PUT ETB. The instruction that moves an ETB into a Communications Facility buffer.

PUT ETX. The instruction that moves an ETX into a Communications Facility buffer.

PUT F. The instruction that moves a data field from an EDX text area into a Communications Facility buffer.

PUT NUL. The instruction that moves a 3270 set buffer address order into a Communications Facility buffer.

PUT Q. The instruction that puts an element onto a queue.

PUT REP. The instruction that moves a 3270 repeat to address order into a Communications Facility buffer.

PUT STX. The instruction that moves an STX into a Communications Facility buffer.

PUT TCB. The instruction that creates a task control block.

PVC (permanent virtual circuit). A permanent virtual connection that provides services similar to a leased line. Data sent to the network through a logical channel being used as a PVC is always delivered to a specific logical channel at a specific DTE destination in the network. In the Communications Facility, PVC circuit stations represent permanent virtual circuits.

PW. The work session controller command that writes data to the priority area of a static screen terminal.

qualifier bit (Q-bit). A bit in a data packet header that indicates the type of information in the packet. Q-bit of 0 (off) means the data is user data; Q-bit of 1 (on) means the data is application-defined control data. Communications Facility application programs send and receive control data in the passthrough control message.

queued message utility ($.UT2). A utility program that allows the user to examine and purge messages on the $.WASTE queue or any other disk queue.

RA. The work session controller command that reads protected and unprotected data from a terminal.

RC. (1) The PD command that sets the number of times the program dispatcher will attempt to load a program when storage is not available. (2) The work session controller command that reads the cursor position from a static screen terminal.

RD. The work session controller command that reads unprotected data from a terminal.

reason code. A code indicating the reason an undeliverable message is in the $.WASTE queue.

reason message. A message preceding each undeliverable message in the $.WASTE queue. The reason message contains the date, time, and reason code.

RECEIVE M. The instruction that receives a message into a Communications Facility buffer.

RECEIVE N. The instruction that determines whether there are any messages on a station's queue.

RECEIVE P. The instruction that purges a message from a station's queue.

RECEIVE T. The instruction that receives a message into an EDX text area.

recognized private operating agency (RPOA). The X.25 facility that defines a particular transit network through which a virtual call is to be routed internationally, when more than one RPOA transit network exists at an international gateway.

Recommendation X.25. The CCITT recommendation that defines standards for the connection of processing equipment to a packet-switching data network. It addresses the physical level, the link level, and the packet level. The Communications Facility adheres to the recommendation as amended in 1981.

remote cell. Any cell in the Communications Facility configuration other than the local cell.

remote node. Any node in the Communications Facility configuration other than the local node.

remote station. A station in a remote node.

reset confirmation control message. A Communications Facility message sent by the X.25 I/O control program to a STD+ circuit station's direct or alternate link informing it that the circuit has been reset as a result of a reset initiated by the application program.

reset indication control message. A Communications Facility message sent by the X.25 I/O control program to a STD+ circuit station's direct or alternate link informing it that the circuit has been reset by the remote data terminal equipment or as a result of an error detected by the network or XHCS.

reset request. A Communications Facility message sent by the STD+ circuit station to reinitialize the virtual call or the permanent virtual circuit. Reinitialization removes all data and interrupt packets in the network.

reverse charging (REV). The X.25 facility used to request that the cost of a communications session be charged to the called data terminal equipment.

RPOA (recognized private operating agency). The X.25 facility that defines a particular transit network through which a virtual call is to be routed internationally, when more than one RPOA transit network exists at an international gateway.

RS. The work session controller command that restores data saved in the $.WSCIMG data set.

RT. The work session controller command that reads the transaction identifiers that are set for the program function keys.

S$CFEQU. The Communications Facility system equate table, which defines names for the fields of the station blocks, blocks, the buffer header, and the message header.

S$GETPNL. The subroutine that fetches panels created by means of the $.PANEL program.

S$POOL. The system storage pool; a workspace pool in the common area, used by the Communications Facility for station blocks and work areas.

SC. The work session controller command that positions the cursor on a static screen terminal.

SD. The work session controller command that starts an EDX terminal as a work session controller terminal.

secondary cell identifier. A field of a transaction, whose meaning is defined by the program that processes the transaction. It may, for example, be the identifier of the cell to which an acknowledgment is to be sent.

secondary transaction identifier. A field of a transaction, whose meaning is defined by the program that processes the transaction. It may, for example, be the identifier of a transaction to be sent as an acknowledgment.

SEND A. The instruction that sends an acknowledgment of receipt of a message.

SEND CP. The instruction that sends a CP command.

SEND E. The instruction that sends an error message to the Communications Facility system log.

SEND L. The instruction that sends an informational message to the Communications Facility system log.

SEND M. The instruction that sends a data message from a Communications Facility buffer.

SEND MT. The instruction that sends a transaction message to the program dispatcher from a Communications Facility buffer.

SEND S. The instruction that sends a status message from an EDX text area. See also *status message*.

SEND SM. The instruction that sends a status message from a Communications Facility buffer.

SEND T. The instruction that sends a message from an EDX text area.

SEND TT. The instruction that sends a transaction message to the program dispatcher from an EDX text area.

SF. The work session controller command that sets the forms parameter for a roll screen or hard copy terminal.

SL. The work session controller command that notifies the work session controller that the terminal's 3270 device station name has changed.

SNA logical unit station. A station that represents an SNA logical unit (a terminal or a printer) in a Communications Facility configuration.

SNA physical unit station. A station that represents an SNA physical unit in a Communications Facility configuration.

SNA prompt screen (panel). An optional menu put out by SNA logical units that prompts the terminal operator to log on to a host SNA application.

SRB (storage resource block). Four words preceding a buffer or workspace acquired from a workspace pool, used to control the allocation of buffers and workspaces from the pool.

SS. The work session controller command that starts a session with a terminal.

ST. (1) The CP command that displays message statistics and Local Communications Controller hardware statistics. (2) The work session controller command that sets the transaction identifiers of the transactions that are to be sent when a PF key on a static screen terminal is pressed after a WK command.

standard field. A protected field of a panel (created through $.PANEL) that has default attributes.

started station. A station that is represented by a station block in S$POOL.

station. A named unit of hardware or software managed by the Communications Facility.

station address. The last two characters of a station's network address. They uniquely identify the station within the node.

station block. A control block in S$POOL that contains information about a started station.

station name. A 1-to-8 character alphameric value that uniquely identifies each station in a node.

station subtype. An attribute of a station that further defines its type; for example, a device-type station may have a subtype such as 3101, 4978, or printer.

station type. An attribute of a station that specifies its type (for example, line, device, terminal, user, or message station).

status message. A message sent with a SEND S or SEND SM command that results in a unique return code (+6) when it is received. A status message is used to (1) tell a station to stop or halt. (2) send X.25 control messages between the X.25 I/O control program and applications linked (alternate or direct) to a STD+ circuit station.

stopped station. A station, represented by a station block in S$POOL, for which the flow of messages has been temporarily stopped. Messages sent to a stopped station are undeliverable.

storage resource block (SRB). Four words preceding a buffer or workspace acquired from a workspace pool, used to control the allocation of buffers and workspaces from the pool.

subtype, station. An attribute of a station that further defines its type; for example, a device-type station may have a subtype such as 3101, 4978, or printer.

SV. The work session controller command that saves data in the $.WSCIMG data set.

SVC (switched virtual circuit). A dynamically-established connection between two pieces of data terminal equipment (DTE). The switched virtual circuit is the packet network equivalent of a switched or dial-up line. In the Communications Facility, SVC circuit stations represent switched virtual circuits.

switched virtual circuit (SVC). A dynamically-established connection between two pieces of data terminal equipment (DTE). The switched virtual circuit is the packet network equivalent of a switched or dial-up line. In the Communications Facility, SVC circuit stations represent switched virtual circuits.

system station. The station block, named $.DISP, that represents the Communications Facility control program.

system storage pool (S$POOL). A workspace pool in the EDX supervisor, used by the Communications Facility for station blocks and work areas.

terminal, Communications Facility. A device defined to the Communications Facility, controlled by an I/O control program, and accessed from a program through SEND and RECEIVE instructions.

terminal, EDX. A terminal defined to the EDX operating system and used to perform EDX system functions.

terminal, work session controller. A terminal managed by the work session controller and accessed from an application program by means of work session controller transactions.

terminal station. A station that represents a 3270 control unit, display, or printer attached to a Series/1, or a station block used to emulate a 3270 control unit, display, or printer in a Communications Facility configuration.

TID (transaction identifier). The 4-character name of a transaction.

TID statement. A statement in $.SYSPD that identifies a transaction to be processed in the local cell.

TN. The work session controller command that sounds the tone on a terminal.

TRAC. The PD command that starts or stops a trace of transactions.

TRAN. The PD command that sends transactions.

transaction. A special-format, user-defined message, routed through the Communications Facility network by the program dispatcher and processed at its destination by a specific transaction-processing program.

transaction identifier (TID). The 4-character name of a transaction.

transaction identifier (TID) table. The table that defines the transactions to be processed in the local cell. It contains, for each transaction, its identifier, its attributes, and the name and attributes of its associated programs.

transaction message. A message, the content of which is a transaction.

transaction table. Synonymous with *transaction identifier (TID) table.*

transaction type. A 2-character indicator of the actions that occur when a transaction is entered: loading one of four types of program, creating a station, and/or sending the transaction message to the station.

transaction-processing program. A program designed to process transactions. The program dispatcher controls loading and execution of transaction-processing programs as it receives transactions.

transparent mode. A mode of BSC transmission that allows any bit pattern to be transmitted as data.

type, line. The combination of station type and station subtype for a communication line. Line type indicates which input/output control program (such as point-to-point or 3270 control) is to control a particular line.

type, message. An attribute of a message that indicates whether it is a data, command, log, transaction, or status message.

type, station. An attribute of a station that specifies its type (for example, line, device, terminal, user, or message station).

type, transaction. A 2-character indicator of the actions that occur when a transaction is entered: loading one of four types of program, creating a station, and/or sending the transaction message to the station.

UK. The work session controller command that unlocks the keyboard on a static screen terminal.

undeliverable message. A message that cannot be delivered because its destination station is stopped or is unknown to the message dispatcher.

unprotected field. A field on a display screen in which the user is allowed to enter data; also, the definition of such a field in a panel or an EDX screen image.

US. The work session controller command that ends a lock sequence for a static screen terminal.

user station. A station that represents a user or system program in a Communications Facility configuration.

vector station. A station block that represents a remote station in a multinode Communications Facility configuration.

virtual call. A temporary logical connection between two pieces of data terminal equipment. Virtual calls are placed through switched virtual circuits. See also *switched virtual circuit.*

virtual circuit. A logical connection established between two pieces of data terminal equipment. It can be permanent—defined when you subscribe to your network port—or it can be switched—dynamically established when a call is placed. The Communications Facility manages stations that represent these circuits; XHCS manages the circuits. See also *switched virtual circuit* and *permanent virtual circuit.*

WD. The work session controller command that writes unprotected data to a terminal.

window. (1) The number of data packets a DTE or DCE can send across a logical channel before waiting for authorization to send another data packet. It is the main mechanism for pacing the flow of X.25 packets across an X.25 network. In the Communications Facility, window is defined in the line or circuit station definition. (2) In $.PANEL, the area on the screen that can be seen one time when defining a 3270 panel.

WK. The work session controller command that waits for a static screen terminal's PF key or ENTER to be pressed or for a timeout set by an IT command to occur.

work session controller ($.WSC). The part of the Communications Facility that allows an application program to communicated with multiple EDX devices attached to any Series/1 in the network.

work session controller data set ($.WSCIMG). The data set that contains images that can be displayed through the work session controller and members used to save data for transaction-processing programs.

work session controller high-level language subroutines. A set of subroutines that allow an EDL or COBOL program access to work session controller functions through subroutine calls rather than through the WSC transaction.

work session controller sample program ($.WSMENU). The program that can be used to start a session between an EDX terminal and work session controller application programs.

work session controller terminal. A terminal managed by the work session controller and accessed from an application program by means of work session controller transactions.

workspace pool. An area of processor storage from which the Communications Facility programs allocate buffers and work areas. The pool includes information used to control the allocation of elements in the pool.

WP. The work session controller command that writes protected data to a static screen terminal.

WSC. (1) The transaction identifier associated with the Communications Facility program $.WSC. (2) The command that starts a work session controller terminal.

X.25 control message. See *control message.*

X.25 data message. See *data message.*

X.25 data set. See *$.SYSX25.*

X.25 header. A header that precedes an X.25 control or data message. It identifies what type of message, control or data, is contained in the message. This header is 2 bytes long in the Communications Facility Version 2.0.

X.25 network. A packet-switching data network that adheres to the standards defined by the CCITT Recommendation X.25.

X.25 network address. A field of up to 15 binary-coded decimal (BCD) digits that identifies the DTE to which a call is directed or from which a call originated. The Communications Facility provides a data set, $.SYSX25, in which the user may relate this address to a two-digit call ID.

XHCS. The IBM Series/1 Event Driven Executive X.25/HDLC Communications Support licensed program (Program Number 5719-HD2). XHCS allows an application program, such as the Communications Facility, to communicate with remote applications through an HDLC communications link using X.25 packet level procedures.

3270 control ($.IO0AC0). The input/output control program that controls 3270 displays and printers attached to the Series/1.

3270 emulation. The facility that allows a host processor to communicate with Series/1 as if it were communicating with a 3270 system; also, the input/output control program that provides 3270 emulation over a BSC line ($.IO0AE0).

3270 panel print utility ($.PNLUT1). The utility program that prints descriptions of panels created by the $.PANEL program.

Index

Special Characters

READER'S COMMENT FORM

SL23-0106-0

IBM Series/1 EDX
Communications Facility
Programmer's Guide

This form may be used to comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers).

IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

If you wish a reply, be sure to include your name and address.

COMMENTS

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.
  FOLD ON TWO LINES, SEAL AND MAIL

Fold and tape

Please do not staple

Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
System Products Division
Dept 4J1/037 - -PAS5
1501 California Avenue
P. O. Box 10500
Palo Alto, CA 94304

Fold and tape

Please do not staple

Fold and tape

READER'S COMMENT FORM

SL23-0106-0

IBM Series/1 EDX
Communications Facility
Programmer's Guide

This form may be used to comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers).

IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

If you wish a reply, be sure to include your name and address.

COMMENTS

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.
  FOLD ON TWO LINES, SEAL AND MAIL

Fold and tape

Please do not staple

Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
System Products Division
Dept 4J1/037 - -PAS5
1501 California Avenue
P. O. Box 10500
Palo Alto, CA 94304

Fold and tape

Please do not staple

Fold and tape

READER'S COMMENT FORM

SL23-0106-0

IBM Series/1 EDX
Communications Facility
Programmer's Guide

This form may be used to comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers).

IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

If you wish a reply, be sure to include your name and address.

COMMENTS

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.
  FOLD ON TWO LINES, SEAL AND MAIL

Fold and tape

Please do not staple

Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
System Products Division
Dept 4J1/037 - -PAS5
1501 California Avenue
P. O. Box 10500
Palo Alto, CA 94304

Fold and tape

Please do not staple

Fold and tape

SL23-0106-0

SL23-0106-0
Program Number:
5719-CF2
File Number: S1-30

Printed in U.S.A.