IBM

Series/1

SC34-0638-0

Event Driven Executive Communications Guide

Version 5.0

Library Guide and Common Index SC34-0645	Installation and System Generation Guide SC34-0646	Operator Commands and Utilities Reference SC34-0644
Language Reference SC34-0643	Communications Guide SC34-0638	Messages and Codes SC34-0636
Operation Guide SC34-0642	Event Driven Language Programming Guide SC34-0637	Reference Cards SBOF-1625
Problem Determination Guide SC34-0639	Customization Guide SC34-0635	Internal Design LY34-0354

IBM

Series/1

SC34-0638-0

Event Driven Executive Communications Guide

Version 5.0

Library Guide and Common Index SC34-0645	Installation and System Generation Guide SC34-0646	Operator Commands and Utilities Reference SC34-0644
Language Reference SC34-0643	Communications Guide SC34-0638	Messages and Codes SC34-0636
Operation Guide SC34-0642	Event Driven Language Programming Guide SC34-0637	Reference Cards SBOF-1625
Problem Determination Guide SC34-0639	Customization Guide SC34-0635	Internal Design LY34-0354

First Edition (December 1984)

This is a revision of, and obsoletes, SC34-0443-0.

Use this publication only for the purposes stated in the following section, "About This Book."

Changes are made periodically to the information herein; any such changes will be reported in subsequent revisions or Technical Newsletters.

This material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Publications are not stocked at the address given below. Requests for copies of IBM publications should be made to your IBM representative or the IBM branch office serving your locality.

This publication could contain technical inaccuracies or typographical errors. A form for readers' comments is provided at the back of this publication. If the form has been removed, address your comments to IBM Corporation, Information Development, Department 28B, P. O. Box 1328, Boca Raton, Florida 33432. IBM may use or 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.

Summary of Changes for Version 5

The only changes made to this document for Version 5 are miscellaneous editorial enhancements.

About This Book

The information in the *Communications Guide* pertains to the Event Driven Executive Version 5, modification level 0. The book describes the use of various forms of data communications that are available with Series/1 and the Event Driven Executive. It covers several methods of Binary Synchronous Communications (BSC), operation of a host system with a Series/1, communications between two Series/1s, and communications between Series/1 and multiple peripheral devices. It tells the reader how to prepare for data communications operations, how to use Event Driven Language (EDL) instructions to perform communications functions, and how to use Event Driven Executive communications utilities.

In conjunction with the *Communications Guide*, the following books provide information to help perform data communications:

- *Language Reference* provides syntax and descriptions of EDL instructions.
- *Messages and Codes* lists the error messages and codes issued for communications.
- *Operator Commands and Utilities Reference* provides information on the use of communications utilities.

Audience

Readers of the *Communications Guide* should have the following:

- Knowledge of data communications concepts in general
- Experience in communications programming for either IBM or non-IBM products

Audience *(continued)*

- Understanding of synchronous and asynchronous line disciplines
- Detailed knowledge of the principles of Binary Synchronous Communications (BSC), including line protocol
- Familiarity with the relationship between a host system and a remote system.

How This Book Is Organized

The book contains three main parts. Each chapter in a part discusses a different method of performing data communications under the Event Driven Executive operating system.

- Part 1, Binary Synchronous Communications, discusses three methods that use forms of BSC protocol.
- Part 2, System/370 Channel Attach, discusses the communication of Series/1 and a large host system over a direct local channel.
- Part 3, Special EDX Communications Methods, discusses two methods unique to Series/1 under the Event Driven Executive operating system.

Aids in Using This Book

This book contains the following aids to using the information it presents:

- A table of contents that lists the major headings in this book.
- An index of the topics covered in this book.
- A glossary that defines terms and acronyms used in this book and in other EDX library publications.

In the step-by-step procedures, several utilities are used and the interactive display screens are shown. Any responses you must make in answer to a prompt are shown in red.

A Guide to the Library

Refer to the *Library Guide and Common Index*, SC34-0645 for information on the design and structure of the Event Driven Executive, Version 5 library, for a bibliography of related publications, and for an index to the entire library.

Contacting IBM about Problems

You can inform IBM of any inaccuracies or problems you find when using this book by completing and mailing the **Reader's Comment Form** provided in the back of the book.

If you have a problem with the Series/1 Event Driven Executive services, fill out an authorized program analysis report (APAR) form as described in the *IBM Series/1 Software Service Guide*, GC34-0099.

Contents

Contents

Contents

Figures

Figures

Introduction to Communications Guide

The *Communications Guide* discusses the data communications methods that are available as part of the Event Driven Executive system, and in conjunction with Installed User Programs (IUPs).

"Part 1. Binary Synchronous Communications" on page CO-3 covers three methods of communications that use binary synchronous protocol:

- Binary Synchronous Communications Access Method (BSCAM)
- Remote Management Utility ($RMU)
- Host Communications Facility (HCF)

"Part 2. Channel Attach" on page CO-143 discusses communications between the Series/1 and a larger host system.

"Part 3. Specialized Series/1 Event Driven Executive Communications Methods" on page CO-173 covers two methods of communications unique to Series/1 under the Event Driven Executive:

- Series/1-to-Series/1 Attachment
- General Purpose Interface Bus (GPIB)

In addition to the methods mentioned above, several licensed programs offer forms of communications with the Event Driven Executive:

- Multiple Terminal Manager
- Systems Network Architecture/Synchronous Data Link Control
- Communications Facility

Licensed programs can be purchased separately from the basic system, along with documentation on their use.

Part 1. Binary Synchronous Communications

Part 1 covers the following forms of binary synchronous communications:

- Binary Synchronous Communications Access Method (BSCAM)
- Remote Management Utility ($RMU)
- Host Communications Facility (HCF)

Notes

Chapter 1. Binary Synchronous Communications Access Method (BSCAM)

The Binary Synchronous Communications Access Method (BSCAM) provides I/O access at a read/write level using a BSC protocol similar to that of BTAM. It allows the Series/1 to communicate with local and remote processors and terminals. The BSCAM access method is also the basis for the Remote Management Utility ($RMU), which runs on the Series/1 and allows it to be controlled by a host system. $RMU is discussed in Chapter 2, "Remote Management Utility ($RMU)" on page CO-57.

BSCAM supports the attachment of multiple BSC lines to the Series/1. Data can be either transparent or nontransparent. Stations acknowledge transmissions either with standard BSC control characters or with conversational responses. The transmission code that BSCAM uses is EBCDIC.

Hardware features and BSC protocol that BSCAM does not support are:

- ASCII mode
- Leading graphics support
- Transparent ITB and ENQ transmission.

BSCAM provides I/O at the READ/WRITE access level. It does not insert or delete control character sequences from the data you place in your buffers. When sending data, you must place the proper start and end-of-transmission control characters (STX, DLE STX, and ETX) in your output buffer. The only case in which you must not place characters in your output buffer is when you are sending the transparent end-of-transmission and end-of-block (DLE ETX and DLE ETB) sequences. Do not place these control characters in your buffer. When you receive data, your input buffer will contain all the control characters that it received in the transmission.

Binary Synchronous Communications Access Method (BSCAM)

For a general introduction to binary synchronous communications and details of line protocol used by the Event Driven Executive , refer to *General Information - Binary Synchronous Communications*, GA27-3004.

Application programs that run under BSCAM consist of special Event Driven Language BSC instructions. These programs send and receive data from one or more stations. In addition, several BSC utilities check the way BSCAM is working by monitoring and simulating its operations.

This chapter tells how to:

- Plan to use BSCAM.
- Write application programs with the special set of BSC instructions.
- Use the BSC utilities to check for potential problems.
- Use the X.21 digital communications capabilities.

Terms Used in this Chapter

Below are some definitions of terms used throughout this chapter.

The computers that are communicating are called *stations*.

The data records sent and received are called *messages*.

The station sending messages (writing) is called the *sending* or *transmitting* station.

The station receiving messages (reading) is called the *receiving* station.

Planning for BSCAM Operations

Before you can begin to use BSCAM, your system must have certain hardware and software support. The combination of support on your system determines the type of programs you should write and the way you can use BSCAM.

The sections that follow will help you to:

- Select the hardware and software you need to use BSCAM and define the support during system generation.
- Determine what hardware and software your system already supports, if system generation has already been done.
- Decide which of the BSCAM features you will use.

Using Data Links

BSCAM supports transmission over switched and nonswitched (leased) data links. The data link is the line provided by a common carrier over which data is transmitted from your station to another station. BSCAM supports half-duplex transmission only.

Only remote connections between stations need a data link. (Local connections use direct-connect.) If you plan to communicate with a remote station, make sure your system is linked to either a switched or a nonswitched line.

Selecting BSC Line Connection Types

One of the most important things you must decide about a BSC line is the type of station it will function as. BSCAM supports connection of multiple BSC lines to a Series/1. Think of each BSC line as a separate physical entity, functioning independently of the others. As far as each BSC line knows, it has an exclusive connection with the Series/1. One BSC line may function as a point-to-point station communicating over a switched data link. Another line (installed on the same processor) may function as a multipoint tributary on a leased link. Still another may be a multipoint control station.

Every BSC line that you have attached to your Series/1 can form one of several types of connection with the processor. Select the type of connection you want to establish for each BSC line on Series/1. Then refer to "Defining a BSC Line to the Supervisor" on page CO-12 for details on defining your selections to the system. If system generation has already been done, use the $IOTEST utility LS (list supervisor configuration) command to determine the assignment of each BSC line.

The type of connection each BSC line forms with your Series/1 affects the way you program that line to send and receive data.

Point-to-Point Connection

If a BSC line forms a point-to-point connection with your Series/1, it will handle communications between your processor and one other station. The connection between the two stations can be either local or remote. In remote connections, the data link can be either switched or leased. Local connections use a direct connection. In a point-to-point connection, each station contends with equal priority for the right to send data across the line.

Planning for BSCAM Operations *(continued)*

Multipoint Connection

If a BSC line forms a multipoint connection with your Series/1, it handles communications between your processor and one or more other stations. You can designate your Series/1 to act either as the control station or as a tributary station in the multipoint connection.

A control station is in charge of which of the multiple stations has the right to transmit at any given time. It governs the other stations by polling (asking the tributary if it is ready to send data) and selecting (asking the tributary if it is ready to receive data). The tributary has direct communication only with the control station, and then only when the control station polls or selects it. To send data to another tributary that is under the same control station, the first tributary sends the data directly to the control station, which in turn sends it to the specified tributary.

For more information on polling and selecting, refer to the section "Sending Poll/Select Sequences" on page CO-20.

Note: The hardware does not use the correct selection sequence if you specify cluster addresses 'C1' and '50' for a device emulating a 3270 on a BSC line.

Meeting Hardware Requirements

Before you can actually use BSCAM to communicate with other stations, you must ensure that the following hardware is installed on your Series/1:

- BSC lines are attached with control provided for each of them.
- Modem or modem eliminator is attached to the BSC line(s) (if applicable).

The attachment (physical connection) and control (interface with BSCAM) of a BSC line can be provided by one of several Series/1 communications features. These features are actually hardware cards installed in the processor or I/O expansion unit of your Series/1. Some types of cards attach a single BSC line to the Series/1, while other cards can attach multiple lines.

Determining Existing Hardware Configuration and BSC Line Addresses

To determine the hardware address and feature type of each communications card already installed on your system, load the $IOTEST utility, and issue the LD (List Devices) command. To determine the address of each BSC line defined to the supervisor, again use $IOTEST and issue the LS (List Supervisor Configuration) command.

Series/1 Communications Features

Here is a list of the communications features that control and attach BSC lines to the Series/1. For details on the functional characteristics and installation of these features, refer to the *IBM Series/1 Binary Synchronous Communications Feature Description*, GA34-0244.

IBM 2074 BSC Single-Line Control, Medium Speed: This communications feature card makes the physical connection of one BSC line to the Series/1. It also provides control of the line. You can use this feature for BSC lines that you will use to form either point-to-point or multipoint connections. The point-to-point connections can be either local or remote; for remote operation, the data link can be either switched or leased. Multipoint connections must use a leased data link. For details on the functional characteristics and installation of this card, refer to the *IBM Series/1 Binary Synchronous Communications Feature Description*, GA34-0244.

The single-line control feature requires an electrical interface compatible with RS-232C (CCITT V.24). Its maximum data transfer rate is 9600 bits per second. In addition, it supports the following calling features: manual call, manual answer, and automatic answer (the last feature applies to switched connections only). The single-line control allows IPL from another system through use of its BSC line.

IBM 2075 BSC Single-Line Control, High Speed: This communications feature card attaches and controls one BSC line. Its characteristics are similar to the medium-speed control (2074) described above, but with the following differences:

- It is for use in remote operations only, since it has no internal clocking and, therefore, cannot be used in direct connections.
- It operates in point-to-point leased and multipoint connections.
- It requires a Western Electric 303 modem (or equivalent), or an electrical interface compatible with CCITT V.35.
- Its maximum data transfer rate is 56K bits per second.

For details on the functional characteristics and installation of this card, refer to the *IBM Series/1 Binary Synchronous Communications Feature Description*, GA34-0244.

Planning for BSCAM Operations *(continued)*

IBM 2080 Synchronous Communications Single-Line Control, High Speed: This communications feature card attaches and controls one BSC line. It is required if you want X.21 digital data communications capabilities for your Series/1. It has the following characteristics:

- It operates in point-to-point connections only for BSC switched mode and point-to-point and multipoint for BSC leased mode.
- It can be jumpered for switched or leased mode.
- It requires an electrical interface compatible with CCITT V.35 (leased) or an X.21 electrical interface (switched or leased).
- Its maximum data transfer rate is 48K bits per second.

For details on the functional characteristics and installation of this card, refer to the *IBM Series/1 Synchronous Communication Single-Line Control Attachment Feature Description*, GA34-0241.

IBM 2093/2094 BSC 8-Line Control and BSC 4-Line Adapter(s): These features, used together, can attach and control up to 8 BSC lines. The 8-line control is for use with one or two 4-line adapters (which provide only attachment of lines). The characteristics of these features are similar to those of the BSC Single-Line Control.

For details on the functional characteristics and installation of this card, refer to the *IBM Series/1 Binary Synchronous Communications Feature Description*, GA34-0244.

IBM 1310 Multifunction Attachment: This feature can attach one BSC line to the Series/1 and conforms to RS-232C interface standards. The use of this feature requires special system definitions during system generation for your Series/1.

For details on the functional characteristics and installation of this card, refer to the *IBM Series/1 Multifunction Attachment Feature and 4975 Printer Description*, GA34-0144.

IBM RPQ D02349 Direct BSC Attachment: This feature controls the serial transfer of data to and from remote terminals or systems using direct-connect cabling for half-duplex, single-line capability. It has the following characteristics (for more information, refer to *IBM Series/1 Direct Binary Synchronous Communication Attachment RPQ D02349 Custom Feature,* GA34-1577:

- It conforms to EIA RS-422 electrical interface standards.
- Data transmission is serial-by-bit using BSC transmission method.
- It can be used as either a primary or a secondary station.
- It is able to send and receive transparent data for EBCDIC only.
- Its maximum data transfer rate is 38,400 bits per second if you jumper for internal clocking and 56,000 bits per second if you jumper for data terminal equipment clocking to another system.

Special Considerations for Multipoint Tributary Stations

If you plan to use a BSC line as a multipoint tributary station, you must ensure that its feature card is jumpered with DTR (data terminal ready) permanently enabled. This means that the electrical interface or modem always leaves the line open to receiving data from its multipoint control station.

Special Considerations for Local (Direct Connect) Operation

If you are using a Multifunction Attachment card to attach a BSC line to your Series/1, and you plan to use that line in a local (direct) connection, then you must use a *modem eliminator*, to act as the electrical interface in the direct connection. The modem eliminator must meet the electrical interface requirements of the feature cards to which it connects.

If you are using a single-line, medium speed control or the 8-line control with 4-line adapter, then you do not need to use a modem eliminator. The direct connection is made between the communicating lines. However, ensure that internal clocking is jumpered on the card(s) to provide direct connection.

As stated previously, the 2075 single-line control, high speed feature card cannot be used for direct connections.

Optional Hardware

The Communications Indicator Panel (model #2000) is an optional piece of hardware that is useful in program debugging and hardware troubleshooting.

Refer to the section "Monitoring BSC Lines with the Communications Indicator Panel" on page CO-45 for information on using the panel.

Planning for BSCAM Operations *(continued)*

Including BSCAM Support in the Supervisor

You must ensure that your supervisor supports the BSC hardware and operation of the BSCAM access method. This is done during system generation, when you must include a statement to define each BSC line to the supervisor. You must also include the supervisor module that supports BSCAM.

Note: For X.21, you must include the BSCX21 module.

The statement that defines BSC lines is called BSCLINE. The supervisor module that includes BSCAM support is called BSCAM. For step-by step directions on performing system generation, refer to the *Installation and System Generation Guide.*

If system generation is already complete, you can find out what the supervisor supports. Load the $IOTEST utility and issue the LS (List Supervisor Configuration) command.

Defining a BSC Line to the Supervisor

Each BSC line attached to the Series/1 needs a BSCLINE statement to define it to the supervisor. Code the statements in the system definition data set, which defines hardware support. For each line, you must define the following:

- Hardware address (in hexadecimal) of the line
- Type of connection (point-to-point, multipoint) it is part of
- Number of retries before a timeout occurs on the line
- Communications feature that attaches the line.

Specifying BSC Line Type (TYPE=Operand of BSCLINE): The TYPE= operand of the BSCLINE statement is where you tell the system what type of station the line functions as. Here is a list of the available station types and how to code them in a BSCLINE statement.

Note: For a complete mapping of connection types for X.21, see "Using X.21 Switched Network Support" on page CO-48.

Type	Connection	What to code
PT	point-to-point local remote (leased)	BSCLINE TYPE=PT
SM	point-to-point remote (switched) manual call	BSCLINE TYPE=SM
SA	point-to-point remote (switched) automatic answer	BSCLINE TYPE=SA
MC	multipoint control station remote (switched)	BSCLINE TYPE=MC
MT	multipoint tributary station remote (switched)	BSCLINE TYPE=MT
For X.21 only		
AC	point-to-point switched auto call	BSCLINE TYPE=AC
DC	point-to-point switched direct call	BSCLINE TYPE=DC

Figure 1. Specifying BSCLINE TYPE= Operand

When a BSC line is attached with a Multifunction Adapter (MFA), you must code the parameter ADAPTER=MFA in the BSCLINE statement for the line. In addition, you must code an ADAPTER statement, specifying the address of the line in the ADDRESS= parameter.

Note: The address specified in the BSCLINE and ADAPTER statements must be identical. Refer to the *Installation and System Generation Guide* for details on the BSCLINE and ADAPTER statements.

When a BSC line acts as a Multipoint Tributary Station, you must specify the poll/select addresses (up to 4) that the line should respond to during polling/selection by its control station.

Planning for BSCAM Operations *(continued)*

Defining BSCAM Supervisor Module Support

You must tell the supervisor to support BSCAM operations. During system generation, edit the link control data set, which defines software support to the supervisor, to include the BSCAM module.

Note: For X.21, you must include the BSCX21 module.

Refer to the *Installation and System Generation Guide* for details on defining BSCAM software support.

Selecting Type of Data to Transmit

You can transmit data in several forms during BSCAM operations. The type of data affects the way you write programs to send it.

Standard Data (Nontransparent)

When BSCAM transmits nontransparent data, the receiving station interprets the bits according to the EBCDIC code. If the data contains bit combinations that represent BSC control characters, the receiving station will recognize and act on the meaning of the characters. Nontransparent data is commonly used to transmit text. You must be sure that the data you wish to send does not contain bits that look like BSC control characters. If the data does contain control characters, the receiving station will not receive the transmission correctly.

Transparent Data

Transparency allows you to transmit data with bit combinations that look like BSC control characters, without the receiving station interpreting them. It allows you to send raw binary data regardless of what the data looks like. It also allows you to store control character sequences in a buffer at the receiving station.

The data-link-escape (DLE) is the control character used to transmit transparent data. The sequence DLE STX tells the station that is going to receive transparent data, and to ignore any control characters in the data. The sequence DLE ETX or DLE ETB signals the end of a transmission of transparent data, and tells the station to begin recognizing control characters again.

While receiving transparent data, a station will only recognize control character sequences preceded by the DLE.

Sending Transparent Data in Blocks

You may want to break up a long transmission of transparent data into blocks and send each separately. Each block of data is checked for transmission errors, and only those blocks not properly received must be sent again. This is more efficient than sending all the data in one very long transmission. In that case, if one error occurred, then the entire body of data would have to be retransmitted, wasting time and tying up resources.

Selecting Mode of Transmission

Transmissions are possible in two different modes under BSCAM.

Standard Transmission Mode

With standard transmission under BSCAM, acknowledgements and responses between stations consist of predefined BSC control characters which are not stored at the receiving station.

Limited Conversational Response Transmission Mode

Under BSCAM, you can transmit control characters or text data in response to a message by using limited conversational mode of transmission. You can send a conversational response only as a positive acknowledgement to a complete message (one that ended with ETX or DLE ETX). You cannot send conversational replies when receiving block messages (ending with DLE ETB or ETB).

You can begin your conversational reply with either SOH or STX, which the other station interprets as a positive acknowledgement to its last transmission. You can also send transparent data (beginning with DLE STX) as a conversational response. However, after you send one transparent conversational response, the next response you send cannot be transparent as well. Conversational responses are stored in the input buffer of the receiving station. BSCAM checks buffer contents and reports any transmission errors that the response may indicate. The first station can send its next message only after it gets the proper conversational response.

Binary Synchronous Communications Access Method (BSCAM)

Programming for BSCAM Applications

To perform BSCAM communications between your Series/1 and any station(s) connected to it, you write application programs using a set of Event Driven Language (EDL) BSC instructions. Listed below are the BSC instructions and their basic uses.

Refer to the *Language Reference* for details on syntax and operands for each of the BSC instructions.

Instruction	Function	Comments
BSCCLOSE	Frees a BSC line for use by other tasks.	Code at the end of each task or program.
BSCIOCB	Specifies BSC line address and buffers for all BSC operations	A nonexecutable instruction, referred to in every other BSC instruction.
BSCOPEN	Prepares a BSC line for use by a task.	Code at the beginning of each program or task.
BSCREAD	Reads data from a BSC line. Consists of several variations, called "types," each used to read data in a different way.	Code to receive data from another station.
BSCWRITE	Writes data to a BSC line. Consists of several variations, called "types," each used to write data in a different way.	Code to initiate data transfer to another station.

Figure 2. Event Driven Language BSC Instructions

Each instruction requires certain BSC control character sequences to be transmitted between stations. The sections that follow, which discuss the use of the BSC instructions, also contain information on the control characters associated with each instruction.

Basic Programming Functions for BSCAM

The two basic BSCAM functions are:

- Write operations to send data to another station
- Read operations to receive data from another station.

In addition, your program must acquire the use of a BSC line and must provide buffers and include other control information.

This section shows how to use the BSC instructions to perform BSCAM functions.

Acquiring Use of a BSC Line

You must gain the exclusive use of a BSC line before beginning a read or write operation, and release it when the operation is over. The BSCOPEN instruction gets the line, and the BSCCLOSE instruction gives up the line. Both instructions require the label of the BSCIOCB instruction associated with the particular operation. The BSCIOCB instruction is discussed in the next section.

```
OPEN  BSCOPEN     BSCIOCB,ERROR=END
CLOSE BSCCLOSE    BSCIOCB,ERROR=END
```

Figure 3. Example of Coding BSCOPEN and BSCCLOSE Instructions

Coding Control Block for Read and Write Operations

BSCIOCB provides control information used by the other BSC instructions to perform read and write operations. Each BSC instruction must refer to the label of a BSCIOCB instruction.

When you code BSCIOCB, specify the following:

- BSC line address to be used throughout the operation
- Address and length of any buffer(s) to be used
- Polling or selection sequence to be used by a control station.

When you code BSCIOCB, you need to know:

- BSC line type (control, tributary, point-to point) in use
- Write type you are using (determines buffer requirements)
- Read type you are using (determines buffer requirements)
- Length of records (messages) to be sent or received
- Tributary station device type (determines poll/select sequence used by control station). Consult the tributary device description manual for correct poll and select sequences.

Below is an example of a BSCIOCB instruction that specifies line 19, and one buffer 80 bytes in length.

```
IOCB   BSCIOCB      19,BUFFER,80
```

Figure 4. Example of Coding BSCIOCB Instruction

Code the BSCIOCB instruction outside the executable area of your programs.

Note: For an X.21 example, see "Using X.21 Switched Network Support" on page CO-48.

Specifying Buffers

The number of buffers required during a BSCAM operation depends on the read or write type.

Most write types need at least one buffer. However, the conversational writes need two buffers, and the types D, E, EX, and N, do not need any buffers at all.

Write type (code)	Number of buffers	Write type (name)
C	1	Continue
CV	2	Continue Conversational
CVX	2	Continue Conversational Transparent
CX	1	Continue Transparent
CXB	1	Continue Transparent Block
D	0	Delay
E	0	End
EX	0	End Transparent
I	1	Initial
IV	2	Initial Conversational
IVX	2	Initial Conversational Transparent
IX	1	Initial Transparent
IXB	1	Initial Transparent Block
N	0	NAK
Q	1	Inquiry
U	1	User
UX	2	User Transparent

Figure 5. Buffers Required for Write Operations

Most read types need only one buffer. However, the types D and Q do not require any buffers.

Read type (code)	Number of buffers	Read type (name)
C	1	Continue
D	0	Delay
E	1	End
I	1	Initial
P	1	Poll
Q	0	Inquiry
R	1	Repeat
U	1	User

Figure 6. Buffers Required for Read Operations

Sending Data

Under BSCAM, the instruction that sends data is BSCWRITE. Control character sequences are generated when a BSCWRITE instruction executes, and certain acknowledgements are exchanged between the sending and receiving stations. You must specify which type of write operation you need to perform, according to the situation. These write types, and the reasons for using each, are covered in the sections that follow.

In your program to send data, you will:

- Define line address and buffers (BSCIOCB).
- Acquire the BSC line for use by your program (BSCOPEN).
- Transmit data with one or more BSCWRITE instructions (one instruction for each message).
- Give up the use of the BSC line (BSCCLOSE).

Selecting BSCWRITE Types

There are several different types of write operation, each used to transmit data in a different form or situation, and each coded a different way with the BSCWRITE instruction. The factors that determine which type to use are:

- Type of data (transparent, nontransparent)
- Mode of transmission (standard, conversational)
- Order of transmission of messages
- Special conditions that occur during transmission.

The sections that follow tell how and when to code the various types of BSCWRITE instruction.

Sending the First Message

Code an *initial write* instruction to send the first message in a transmission. Different types of initial write instructions exist for each of the modes of transmission, for each of the data types, and for combinations of the two.

In the following table, find the type of data you are sending and the mode of transmission you're using. The table will tell you which type of initial write instruction to code, and how to code it. Each write type consists of BSCWRITE followed by a suffix.

Data Type	Transmission Mode	Write Type	What to code
Standard	Standard	Initial	BSCWRITE I
Standard	Conversational	Initial Conversational	BSCWRITE IV
Transparent	Standard	Initial Transparent	BSCWRITE IX
Transparent	Conversational Transparent	Initial Conversational Transparent	BSCWRITE IVX
Transparent Block	Standard	Initial Transparent Block	BSCWRITE IXB

Figure 7. Initial Write Types

Sending Poll/Select Sequences

If your station is acting as the control station on a multipoint line, then it must poll and select its tributaries. To send each poll/select sequence, code any initial write instruction. The poll/select sequence consists of the following characters:

EOT (poll or selection address) ENQ--------->

You must place the ENQ and the poll or selection address in your output buffer, as specified in your BSCIOCB statement. For more information on the poll/select sequence, refer to *General Information - Binary Synchronous Communications*.

Control Characters Associated with Initial Write Instructions

When an initial write instruction executes, certain BSC control characters must go across the line to the receiving station. The tables that follow show the flow of characters that must accompany each initial write instruction. The access method generates the ENQ and ACK sequences, but you must supply all other control characters in your output buffer.

Instruction	Sending Station Type of Connection	Control Character Flow Sending	Receiving
BSCWRITE I	point-to-point	ENQ---------->	
			<----------ACK
		ETX (Text) STX--->	
			<----(Response character)
BSCWRITE I	multipoint control station	EOT---------->	
		ENQ (Address)------->	
			<----------ACK
		ETX (Text) STX------>	
			<-----(Response character)
BSCWRITE I	multipoint tributary	ETX (Text) STX----->	
			<-----(Response character)

Figure 8. Control Character Flow for Initial Write Instructions

Instruction	Sending Station Type of Connection	Control Character Flow Sending	Receiving
BSCWRITE IV	point-to-point	ENQ---------->	
			<----------ACK
		ETX (Text) STX----->	
			<----(Response Text)
BSCWRITE IV	multipoint control station	EOT---------->	
		ENQ (Address)----->	
			<----------ACK
		ETX (Text) STX---->	
			<-----(Response Text)
BSCWRITE IV	multipoint tributary	ETX (Text) STX----->	
			<-----(Response Text)

Figure 9. Control Character Flow for Initial Conversational Write Instructions

Instruction	Sending Station Type of Connection	Control Character Flow Sending	Receiving
BSCWRITE IX	point-to-point	ENQ---------->	
			<----------ACK
		DLE ETX (Text) DLE STX----->	
			<-----(Response Character)
BSCWRITE IX	multipoint control station	EOT---------->	
		ENQ (Address)------>	
			<----------ACK
		DLE ETX (Text) DLE STX----->	
			<-----(Response Character)
BSCWRITE IX	multipoint tributary	DLE ETX (Text) DLE STX----->	
			<-----(Response Character)

Figure 10. Control Character Flow for Initial Transparent Write Instructions

Instruction	Sending Station Type of Connection	Control Character Flow Sending	Receiving
BSCWRITE IXB	point-to-point	ENQ---------->	
			<----------ACK
		DLE ETB (Text) DLE STX----->	
			<----(Response Character)
BSCWRITE IXB	multipoint control station	EOT---------->	
		ENQ (Address)----->	
			<----------ACK
		DLE ETB (Text) DLE STX----->	
			<-----(Response Character)
BSCWRITE IXB	multipoint tributary	DLE ETB (Text) DLE STX----->	
			<-----(Response Character)

Figure 11. Control Character Flow for Initial Transparent Block Write Instructions

Instruction	Sending Station Type of Connection	Control Character Flow Sending	Receiving
BSCWRITE IVX	point-to-point	ENQ---------->	
			<----------ACK
		DLE ETX (Text) DLE STX----->	
			<-----(Response Text)
BSCWRITE IVX	multipoint control station	EOT---------->	
		ENQ (Address)----->	
			<----------ACK
		DLE ETX (Text) DLE STX----->	
			<----(Response Text)
BSCWRITE IVX	multipoint tributary	DLE ETX (Text) DLE STX----->	
			<-----(Response Text)

Figure 12. Control Character Flow for Initial Conversational Transparent Write Instructions

Sending Subsequent Messages

Code a *continue write* to send the second message and any further messages in a transmission. Code a continue write only after coding an initial write. Also, always make sure that the continue write type matches the initial write type (standard, conversational, transparent, transparent block, or conversational-transparent type). Refer to the table that follows to see which continue write type to code.

Data Type	Transmission Mode	Write Type	What to Code
Standard	Standard	Continue	BSCWRITE C
Standard	Conversational	Continue Conversational	BSCWRITE CV
Transparent	Standard	Continue Transparent	BSCWRITE CX
Transparent	Conversational	Continue Conversational Transparent	BSCWRITE CVX
Transparent Block	Standard	Continue Transparent Block	BSCWRITE CXB

Figure 13. Continue Write Types

Control Characters Associated with Continue Write Instructions

When a continue write instruction executes, certain BSC control characters must be sent to the receiving station. The tables that follow show the control character flow for each continue write type.

Instruction	Sending Station Type of Connection	Control Character Flow Sending	Receiving
BSCWRITE C	all types	ETX (message 1+n text) STX--->	<-----Response character
BSCWRITE CV	all types	ETX (message 1+n text) STX--->	<-----Response Text
BSCWRITE CX	all types	DLE ETX (message 1+n text) DLE STX--->	<-----Response character
BSCWRITE CXB	all types	DLE ETB (Message 1+n Text) DLE STX--->	<-----Response Character
BSCWRITE CVX	all types	DLE ETX (Message 1+n text) DLE STX--->	<-----Response Text

Figure 14. Control Character Flow for Continue Write Instructions

Coding Special Write Types

Besides sending messages, you may need to control data transmissions in special ways during a write operation. The write types listed below cause BSCAM to perform specific functions. You will need to determine where and when these functions are useful in your program.

Write Type	What to Code	Explanation
Delay	BSCWRITE D	Delays transmission of next message. You can code multiple delays before transmission resumes.
Inquiry	BSCWRITE Q	Requests retransmission of response (text or acknowledgement sequence) to a previously sent message.
NAK	BSCWRITE N	Transmits the Negative Acknowledgement (NAK) character. A tributary can write NAK if not ready during polling/selection.
User	BSCWRITE U	Transmits a character stream.
User Transparent	BSCWRITE UX	Transmits character stream in transparent mode.

Figure 15. Special Write Types

Delaying Transmission

You can inform the receiving station that transmission of the next message will be delayed with a *delay write* instruction. You can code multiple delays before resuming transmission.

Sending a Negative Acknowledgement

When you need to transmit a negative acknowledgement (NAK) character, code a *NAK write.* This instruction simply sends the NAK character. Tributary stations can send it to signify "device not ready" in response to polling or selection by the control station.

Sending an ENQ Character

To send an ENQ character, code an *inquiry write* instruction. This instruction is most commonly used to request retransmission of the receiving station's response to the last message sent.

Sending a Data Stream

In special situations you may want to send a data stream, generating no control characters. This is possible with either a *user write* (to send standard data) or a *user transparent write* (to send transparent data).

Unlike other write instructions, the user and user transparent types do not generate the transmission of any predefined control characters or require any acknowledgement from the receiving station. In addition, BSCAM does not perform error recovery for the data stream. The data stream consists of the contents of the first buffer you specify in a BSCIOCB instruction. With the user write instruction, once the contents of the buffer are sent, the operation is over. However, with the user transparent write, you must signal the end of the operation by transmitting the contents of a second buffer specified in BSCIOCB. The buffer can contain any of these control character pairs: DLE ETX, DLE ETB, or DLE ENQ.

Control Characters Associated with Special Write Instructions

The following control characters are sent by the special write instructions.

Instruction	Sending Station Type of Connection	Control Character Flow Sending	Receiving
BSCWRITE D	all types	TTD---------->	<----------NAK
BSCWRITE Q	all types	ENQ---------->	<----Response Character or Text
BSCWRITE N	all types	NAK---------->	

Figure 16. Control Character Flow for Special Write Type Instructions

Ending a Write Operation

Once you have written all your data, you need to tell the other station not to expect any more. Two write types signify the end of a write operation.

The *end write* instruction (BSCWRITE E) sends an EOT to end write operations in any form of transmission.

The *end transparent write* instruction (BSCWRITE EX) sends a DLE EOT character to indicate the end of a write operation. Use this instruction if you are transmitting over a switched line.

Programming Sequence for Write Operations

Now that you know about the various write types, you'll want to be sure to code them in the right order. You'll also want to make sure that all the write types "match up" in your program.

Refer to the chart below and find the mode of transmission you plan to use in your program. It shows the basic sequence of write instructions to use.

Mode of Transmission, Data Type	Writing First Message	Writing Next Message	Optional	Ending Write Operation
Standard, Standard data	I	C	D,Q,U,N	E or EX
Conversational Standard data	IV	CV	D,Q,U,N	E or EX
Standard, Transparent data	IX	CX	D,Q,U,N	E or EX
Standard, Transparent Block data	IXB	CXB	D,Q,U,N	E or EX
Conversational, Transparent data	IVX	CVX	D,Q,U,N	E or EX

Figure 17. Programming Sequences for Sending Data

Receiving Data

A *read* operation retrieves data from a BSC line. Each time another station sends data to your Series/1, you must have a read operation programmed to receive it.

In your read program you will do the following:

- Define line address and buffers (BSCIOCB)
- Prepare the BSC line for use by your program (BSCOPEN)
- Receive data with one or more BSCREAD instructions (one instruction for each message)
- Give up the use of the BSC line (BSCCLOSE).

Selecting BSCREAD Types

Several different types of read operation are available. Each is associated with a different BSCREAD instruction. Each type is used in certain situations, similar to the way the various write types are used. However, with read types it doesn't matter what mode of transmission you are using. With read types, it is *when* you issue the instruction that is important. For example, will the instruction read the first message? Or will it read the second message, or the last message? These are some of the questions you must ask when preparing to code read types.

The chart on the next page lists all the read types, and how and when to code them. They are in alphabetical order, not in the order you would code them in a program.

Read Type	What to Code	Explanation
Continue	BSCREAD C	Reads subsequent message after first message read by Read Initial. Issue until EOT received.
Delay	BSCREAD D	Acknowledges correct receipt of message. Requests sender to wait before sending the next message. Multiple delays can be issued before resuming transmission.
End (Reverse Interrupt)	BSCREAD E	Positive acknowledgment that requests sender to end current transmission and reverse the direction of data transfer.
Initial	BSCREAD I	Reads first message in a transmission.
Poll	BSCREAD P	Reads the poll/select sequence sent by a control station to a Series/1 that is a tributary on a multipoint line.
Inquiry	BSCREAD Q	Reads the ENQ character.
Repeat	BSCREAD R	Requests retransmission of the last message sent.
User	BSCREAD U	Reads data stream. Does not perform error recovery.

Figure 18. Read Types

Receiving the First Message

Regardless of its mode of transmission, you will always read the first message with an *initial read* instruction, *BSCREAD I.*

Receiving Subsequent Messages

To read subsequent messages, code a *continue read*, *BSCREAD C.* Issue continue read instructions until the transmitting station sends the end-of-transmission (EOT) sequence.

Delaying Transmission of Data

After successfully reading a message you may wish the sender to pause before sending the next one. A *delay read* instruction, *BSCREAD D*, causes this pause. You can issue as many delays as

needed before telling the sender to resume transmission. (After the last delay, transmission resumes automatically.)

Responding to a Poll/Select Sequence

If your Series/1 is a tributary station on a multipoint line, code a *poll read* (BSCREAD P) instruction to receive polling and selection sequences from the control station. Once your station is polled or selected, it should issue the appropriate read or write initial instruction.

Reading an ENQ Character

To read an ENQ from the sending station, code an *inquiry read* instruction, *BSCREAD Q*.

Requesting Repeat of a Message

If you are unsuccessful in reading the last message that was sent, you can ask the sender to retransmit it with a *repeat read* instruction, *BSCREAD R*.

Reading a Data Stream

The sender can transmit a data stream, which consists of the contents of a buffer. To receive this data, code a *user read* instruction, *BSCREAD U*. BSCAM does not perform error recovery during this type of read operation.

Ending the Read Operation (Reverse Interrupt)

The RVI (reverse interrupt) control sequence is a positive response used in place of ACK 0 or ACK 1. The receiving station transmits an RVI to request the sender to end current transmissions because the receiver now wishes to send. The sending station treats the RVI as a positive acknowledgment and responds by transmitting all data that prevents it from becoming a receiving station. This may require more than one block transmission. For the RVI, code the *end read* instruction, *BSCREAD E*.

Providing for Errors During BSCAM Operations

All the BSC instructions (except BSCIOCB, which is non-executable) allow you to code routines to take over during error or end conditions. BSCOPEN and BSCCLOSE have the ERROR= operand; BSCREAD and BSCWRITE have both ERROR= and END= operands. It is useful to provide for error recovery in situations such as:

- Errors during opening or closing of the BSC line
- Errors in starting the program
- Errors in doing the initial read or write
- Errors in continuing reads or writes
- Errors in ending the read or write operation.

Common routines are to print error messages and BSC return codes in response to errors, or to restart operations at the previous phase or at the beginning.

BSCAM Sample Programs

The following sample programs perform BSCAM operations. Note that both programs include routines that take over when an error occurs during any phase of the operation.

WRITE Sample Program

This program performs a write operations in transparent mode of transmission. The program does the following:

1. Communicates with the READ sample program
2. Opens the BSC line
3. Performs initial write of data
4. Performs calculations to build the next message to send
5. Performs continue writes until all data is sent

6. Ends the write operation

7. Prints error messages and return codes if a failure occurs during any phase of the program

8. Closes the BSC line.

```
WRITEX     PROGRAM       START
START      BSCOPEN       IOCB,ERROR=PRINTERR
RESTART    BSCWRITE      IX,IOCB
*************************************************
OPEN THE LINE AND BEGIN INITIAL TRANPARENT WRITE
*************************************************
           IF            (WRITEX,EQ,10),GOTO,RESTART
           IF            (WRITEX,NE,-1),GOTO,PRINTERR
           DO            29,TIMES
            ADD          I,1
            CONVTB       MSG#,1
*************************************************
CONTINUE THE WRITE OPERATION
*************************************************
            BSCWRITE     CX,IOCB,ERROR=PRINTERR
           ENDDO
*************************************************
END THE WRITE OPERATION
*************************************************
           BSCWRITE      E,IOCB,ERROR=PRINTERR
           GOTO          ALLDONE
*************************************************
ERROR ROUTINE: PRINT ERROR MESSAGE AND BSC RETURN CODE
*************************************************
PRINTERR   MOVE          ERRCODE,WRITEX
           PRINTEXT      'WRITE ERROR:',SKIP=1
           PRINTNUM      ERRCODE
*************************************************
CLOSE THE LINE
*************************************************
ALLDONE    BSCCLOSE      IOCB
           PROGSTOP
*************************************************
CONTROL INFORMATION:  WRITE DATA TO BSC LINE 19
USE A BUFFER LENGTH OF 82 CHARACTERS
*************************************************
IOCB       BSCIOCB       19,BUFFER,82
BUFFER     DC            X'1002'
           DC            CL74'TEST MESSAGE'
MSG#       DC            CL6'     1'
I          DC            F'1'
ERRCODE    DC            F'0'
           ENDPROG
           END
```

READ Sample Program

This program reads the transparent data sent in the preceding WRITE program. The program does the following:

1. Gains use of the system printer
2. Opens the BSC line
3. Performs initial read of data
4. Prints data on printer
5. Reads subsequent data and prints it
6. Prints error messages and return codes if failure occurs during any phase of the program
7. Ends the read operation
8. Closes the BSC line.

BSCAM Sample Programs *(continued)*

```
READX     PROGRAM     START
START     ENQT        $SYSPRTR
          BSCOPEN     IOCB,ERROR=PRINTERR
*************************************************
OPEN THE LINE AND BEGIN INITIAL READ
*************************************************
RESTART   BSCREAD     I,IOCB
          IF          (READX,EQ,10),GOTO,RESTART
          IF          (READX,NE,-1),GOTO,PRINTERR
PRINTIT   MOVE        MSG,INPUT+2,(80,BYTE)
          PRINTEXT    MSG, SKIP=1
*************************************************
CONTINUE THE READ OPERATION
*************************************************
          BSCREAD     C,IOCB,END=ALLDONE,ERROR=PRINTERR
          GOTO        PRINTIT
*************************************************
ERROR ROUTINE:  PRINT ERROR MESSAGE AND BSC
RETURN CODE, HAVE SENDER REPEAT MESSAGE
*************************************************
PRINTERR  MOVE        RETCODE,READX
          PRINTEXT    ERRMSG,SKIP=1
          PRINTNUM    RETCODE
          BSCREAD     R,IOCB,ERROR=ALLDONE,END=ALLDONE
          GOTO        PRINTIT
ALLDONE   DEQT
*************************************************
CLOSE THE LINE
*************************************************
          BSCCLOSE    IOCB
          PROGSTOP
*************************************************
CONTROL INFORMATION: READ FROM BSC LINE 29,
USE A BUFFER 83 CHARACTERS IN LENGTH
*************************************************
IOCB      BSCIOCB     29,INPUT,83
INPUT     DC          CL83'  '
MSG       TEXT        LENGTH=80
ERRMSG    TEXT        'READ ERROR:'
RETCODE   DC          F'0'
          ENDPROG
          END
```

Interacting with BSCAM (Using BSC Utilities)

The BSCAM utilities ($BSCTRCE, $BSCUT1, $BSCUT2) help you check out the way BSCAM is working and point out any problems that may exist. This section shows what the BSC utilities allow you to do, and what types of information you can gather about BSCAM operations.

Tracing I/O Activities on a BSC Line (Using $BSCTRCE)

The $BSCTRCE utility traces activity on the BSC line you specify. The trace information goes into a trace file, which you must allocate before beginning the trace. (The data in the trace file is unformatted. To format the data for a printout or to display it on a screen, use the $BSCUT1 utility.)

Since I/O activity on a BSC line is controlled by an application program, you must have one of your programs loaded and running at the same time you use $BSCTRCE. The program must be the one controlling the line you specify to be traced.

$BSCTRCE writes trace file records at the completion of a BSC operation. Therefore, when testing a conversational write, if you specify the same buffer address for both input and output, the trace file does not show the data that was transmitted; it shows only the conversation responses received.

Multiple BSC lines may be traced concurrently with multiple loads of $BSCTRCE using different trace files. Each copy of $BSCTRCE must use a different trace data set. Each trace data set name should reflect a unique line number.

When $BSCTRCE terminates, it displays the relative record number of the last trace record written.

Allocating the Trace File Data Set

You must allocate the data set that will contain the trace information before using the $BSCTRCE utility. It must be a "data" type data set. You can allocate the data set at any size you wish. Use the $DISKUT1 utility to allocate the data set.

When the end of the output file is reached, it is reused from the beginning, since $BSCTRCE displays the relative record number of the last trace record written upon termination. The trace file can then be displayed or listed using the $BSCUT1 utility.

Invoking $BSCTRCE

You must load $BSCTRCE in the same partition as the application program that is controlling the line you want to trace.

Note: If you want to use the $BSCTRCE utility with X.21, you must load the utility first.

Interacting with BSCAM (Using BSC Utilities) *(continued)*

Use the $L command or option 8.1 of the session manager. When you have loaded $BSCTRCE, it prompts you for the disk or diskette file where you want to place the trace output. $BSCTRCE then prompts you for the line number you want to trace. You can end the trace action with the attention command STOP.

```
> $L $BSCTRCE
LOADING $BSCTRCE    6P,11:03:22, LP=6500, PART=2
DS1(NAME,VOLUME):  TRACE9,MYVOL
ENTER LINE NUMBER (HEX):  9
        .
        .
        .
> STOP

LAST TRACE RECORD EQUALS  19
$BSCTRCE ENDED AT 11:13:31
```

Specifying BSC Line to Trace

In response to the utility's prompt, enter the number of the BSC line you want to trace. Make sure it is the same line that the loaded application program is controlling.

Terminating the Trace

To end the trace at any point, press the attention key and enter STOP. The utility displays the number of the last record it traced.

Tracing Multiple BSC Lines

You can perform traces on multiple lines at the same time. For each trace, do the following:

- Load the application program that controls the line.
- Load $BSCTRCE in the same partition as the program.
- Specify the BSC line number.
- Direct each trace file to a different data set (the volumes can be the same).

Formatting Trace Files for Print or Display (Using $BSCUT1)

Once you have run $BSCTRCE and wish to see what is in the trace file, use the $BSCUT1 utility to format the file and send it to a printer or terminal. You can select the record for the trace file to dump. You will be prompted, as necessary, for information required by the functions of $BSCUT1.

Invoking $BSCUT1

You invoke $BSCUT1 with the $L command or option 8.2 of the session manager.

$BSCUT1 Commands

To display the $BSCUT1 commands at your terminal, enter a question mark in response to the prompting message COMMAND (?):.

```
> $L $BSCUT1
LOADING $BSCUT1    21P,00:04:21, LP= 9200, PART=1

COMMAND(?):  ?

CV - CHANGE VOLUME
DP - PRINT TRACE FILE ON PRINTER
DU - DUMP TRACE FILE ON TERMINAL
     (CA WILL CANCEL)
EN - END PROGRAM

COMMAND (?):
```

After $BSCUT1 displays the commands, it prompts you with COMMAND (?):. Then you can respond with the command of your choice (for example, CV).

Example: Figure 19 shows dumping records in a trace file to the terminal.

```
COMMAND (?):  DU TRACE9
FIRST RECORD:  32
LAST RECORD:  33

DUMP OF TRACE FILE TRACE9 ON EDX002

*****  RECORD  32  *****  START OF CHAINED OPERATION

CC =  0002  ISW =  A009  STATUS =  98DA 0001 C080
RESULT:  EXCEPTION - WRONG LENGTH RECORD (SHORT)

DCB =  8004 0000 0000 0000 0000 2B1C 0002 2AE4
OPERATION:  CHAINED TRANSMIT

DATA LENGTH =     2
1     1061
```

Figure 19. Dumping Trace File Records

The following screen shows the display for the LAST RECORD selected in the previous example, which was record number 33.

```
*****  RECORD  33  *****  CONTINUATION OF CHAINED OPERATION

DCB =  2008 0000 0000 0000 0000 0000 0200 96F6
OPERATION:  RECEIVE WITH TIMEOUT

DATA LENGTH =    485
1   0227 615B F1F6 4BF5 F94B F3F4 40D1 D6C2  |../$16.59.34 JOB|
17  4040 F4F2 F440 D7D9 F3F0 F1F6 F5F6 40C5  |  424 PR301656 E|
33  E7C5 C3E4 E3C9 D5C7 40D4 40D7 D9C9 D640  |XECUTING M PRIO |
49  40F7 1E27 615B F1F6 4BF5 F94B F3F4 40D1  | 7../$16.59.34 J|
65  D6C2 4040 F4F2 F340 C8D8 F1F2 F1F6 F5F6  |OB  423 HQ121656|
81  40C5 E7C5 C3E4 E3C9 D5C7 40D4 40D7 D9C9  | EXECUTING M PRI|
97  D640 40F7 1E27 615B F1F6 4BF5 F94B F3F4  |O  7../$16.59.34|
113 40D1 D6C2 4040 F3F0 F040 C9E2 F0F3 F1F4  | JOB  300 ISO314|
129 F4F5 40C5 E7C5 C3E4 E3C9 D5C7 40E5 40D7  |45 EXECUTING V P|
145 D9C9 D640 40F5 1E27 615B F1F6 4BF5 F94B  |RIO  5../$16.59.|
161 F3F4 40D1 D6C2 1D43 F4F8 407B C7E2 D7C5  |34 JOB..48 #GSPE|
177 F0F1 F040 D6D5 40D7 D9C9 D5E3 D9F2 4040  |010 ON PRINTR2  |
193 D7D9 C9D6 4040 F51E 2761 5BF1 F64B F5F9  |PRIO  5../$16.59|
209 4BF3 F440 D1D6 C240 40F3 F2F0 40C6 C7F6  |.34 JOB  320 FG6|
LAST 4 D4D5 1E26                             |MN..            |

DUMP COMPLETE
ANOTHER AREA?
```

Figure 20. Dump of the LAST RECORD of Trace File

Reading Trace File Records

Trace records can be up to 256 bytes long. They consists of several fields, as shown in Figure 21.

Field	Size (bytes)	Explanation
CC	2	Condition code
ISW	2	Interrupt status word
STATUS	6	Cycle status words: 3 words, 2 bytes each
DCB	16	Device Control Block
LGTH	2	Length of data sent or received
DATA	up to 224	Data in main storage.
LAST4	4	Last four bytes of data if total data is longer than 224 bytes

Figure 21. Trace Record Fields

Notes:

1. The CC, ISW, and STATUS fields are zero when the DCB has been chained from the previous record's DCB.

2. $BSCTRCE always reports an error on a chained DCB to the first DCB in a chain.

Refer to the *IBM Series/1 Communications Features Description*, GA34-0028 for descriptions of the interrupt condition code, interrupt status word, cycle status words, and device control block.

Interacting with BSCAM (Using BSC Utilities) *(continued)*

Testing BSCAM Operations (Using $BSCUT2)

With the $BSCUT2 utility you can test these BSCAM capabilities:

- Read and write operations
  - standard transmission of transparent and standard data
  - conversational transmission of transparent and standard data.
- Polling and selection by control station on a multipoint line
- Response to polling and selection by tributary station on a multipoint line.

The $BSCUT2 utility prompts you for information such as:

- BSC line addresses
- Device addresses of communications feature cards
- Record length, also called buffer size, in bytes
- Number of records to be transmitted or received.

The utility examines the information you supply in response to its prompts. If you supply incorrect information, the utility cannot perform its test and issues an error message.

By examining the information you supply, $BSCUT2 also checks the BSCLINE statements included in the supervisor, and the customized jumper assignments in BSC hardware features. You can use $BSCTRCE to trace the exercising activities of $BSCUT2. You can format and print the records with $BSCUT1.

Hardware Considerations When Using $BSCUT2

You can use $BSCUT2 to test BSCAM on just one Series/1 or between two Series/1's. When testing with just one Series/1, you must have two BSC lines and wrap a connection between them. Assign one line to do the "read" and the other to do the "write."

If you are running a test between two processors (one performing a read, the other a write operation), load $BSCUT2 on both processors and enter one BSC line address at the "read" processor and another BSC line address at the "write" processor. If you specify an invalid address for one of the processors, the test will fail.

Invoking $BSCUT2

You invoke $BSCUT2 with the $L command or option 8.3 of the session manager.

$BSCUT2 Commands

To display the $BSCUT2 commands at your terminal, enter a question mark in response to the prompting message COMMAND (?):.

```
> $L $BSCUT2
LOADING $BSCUT2    76P,00:05:31, LP= 9200, PART=1

COMMAND (?):  ?
RWI ---- READ/WRITE - NONTRANSPARENT
RWIX --- READ/WRITE - TRANSPARENT
RWIXMP - READ/WRITE - MULTIDROP LINE TRANSPARENT
RI ----- READ - TRANSPARENT/NONTRANSPARENT
WI ----- WRITE - NONTRANSPARENT
WIX ---- WRITE - TRANSPARENT
EN ----- END THE PROGRAM
CH ----- CHANGE HARD-COPY DEVICE
RWIVX -- READ/WRITE - TRANSPARENT CONVERSATIONAL
RWIV  -- READ/WRITE - NONTRANSPARENT CONVERSATIONAL
```

After $BSCUT2 displays the commands, it prompts you with COMMAND: (?):. Then you can respond with the command of your choice (for example, RI).

Testing Read and Write Capability Simultaneously

With BSCAM you can test read and write operations at the same time. You can run the tests between two processors, or between two lines on one processor. If you are using one processor, make sure you specify separate lines, one to read and the other to write. The read/write tests available are:

- Read/write of standard data, standard transmission: RWI command
- Read/write of standard data, conversational transmission: RWIV command
- Read/write of transparent data, standard transmission: RWIX command
- Read/write of transparent data, conversational transmission: RWIXV command.

Interacting with BSCAM (Using BSC Utilities) *(continued)*

For these read/write tests, the utility prompts for the following information:

READ ADDRESS and WRITE ADDRESS refer to the device address of the BSC hardware feature. If the test is to be run between two processors (one to read and one to write), load $BSCUT2 on both processors and enter the correct address for read on one processor and the correct address for write on the other processor. One of the addresses can be invalid and the task for the invalid address on each processor will fail due to an undefined line. However, the read/write task will function properly. This is true for all $BSCUT2 commands.

The RECL prompts refer to the buffer size to be used and, therefore, the number of bytes transferred in one transmission over the BSC line. The maximum buffer size permitted is 512 bytes. READ (RECL) should always be equal to or greater than WRITE (RECL) or errors will occur.

NUMBER OF RECORDS determines the number of transmissions to be made before the test ends.

The MONITOR function causes each task to report its progress to the terminal. If the monitor function is enabled, messages such as TASK ENTERED and TASK EXITED are written to the terminal.

Example: RWI command

```
COMMAND (?):  RWI
RWI ---- READ/WRITE - NONTRANSPARENT
READ ADDRESS?  5A
WRITE ADDRESS?  5B
READ RECL?  80
WRITE RECL?  80
NUMBER OF RECORDS?  10
READ MONITOR?  Y
WRITE MONITOR?  Y
```

Example: RWIV command

```
COMMAND (?):  RWIV
RWIV --- READ/WRITE - NONTRANSPARENT CONVERSATIONAL
READ ADDRESS?  5B
WRITE ADDRESS?  5A
BUFFER LENGTH?  80
NUMBER OF RECORDS?  5
READ MONITOR?  Y
WRITE MONITOR?  Y
```

Example: RWIVX command

```
COMMAND (?):  RWIVX
RWIVX -- READ/WRITE - TRANSPARENT CONVERSATIONAL
READ ADDRESS?  5A
WRITE ADDRESS?  5B
BUFFER LENGTH?  5
NUMBER OF RECORDS?  10
READ MONITOR?  Y
WRITE MONITOR?  Y
```

Example: RWIX command

```
COMMAND (?):  RWIX
RWIX --- READ/WRITE - TRANSPARENT
READ ADDRESS?  5A
WRITE ADDRESS?  5B
READ RECL?  80
WRITE RECL?  80
NUMBER OF RECORDS?  10
READ MONITOR?  Y
WRITE MONITOR?  Y
```

Testing Read and Write on a Multipoint Line (RWIXMP)

This command tests the polling/selection capabilities between stations and also reads and writes transparent data. One BSC line acts as the control station, and one or more other stations act as tributary stations.

When you issue the RWIXMP command, you must answer prompts for:

- Control station device address
- Number of tributaries
- Tributary station device address and tributary address
- Loop count for the control station
- Buffer length and number of records to be exchanged.

The control station device address refers to the address of the BSC hardware feature. The tributary station address refers to the jumpered tributary address on each hardware feature card. You must jumper the adapter in tributary mode for this test to function properly. Loop count refers to number of times you want to send a block of messages.

To perform this test with $BSCUT2 running on two processors:

- Processor 1 uses a valid control station (MC) address and dummy tributary (MT) addresses. It acts as the control station.
- Processor 2 uses a dummy control station address and valid tributary (MT) addresses. It acts as the tributary station.
- Specify the same number of tributaries on both processors.
- Specify the same loop count on both processors.

Example: RWIXMP command

```
COMMAND (?):  RWIXMP
RWIXMP - READ/WRITE - MULTIDROP LINE TRANSPARENT
MC DEVICE ADDRESS?  50
BUFFER LENGTH?  80
NUMBER OF RECORDS?  5
LOOP COUNT?  1
MONITOR?  Y
NUMBER OF TRIBUTARIES?  1

PARAMETERS FOR TRIBUTARY?  1
MT DEVICE ADDRESS?  51
MT TRIBUTARY ADDRESS?  02
BUFFER LENGTH?  80
NUMBER OF RECORDS?  5
MONITOR?  Y
```

DEVICE ADDRESS for this command refers to the device address of the BSC hardware feature. TRIBUTARY ADDRESS refers to the jumpered tributary address on each hardware feature card. LOOP COUNT refers to the number of times $BSCUT2 sends the block of messages that you have specified.

Note: The adapter must be jumpered in tributary mode for this test to function properly.

Testing Read Capability

The RI command tests the read capability of both standard and transparent data. You don't have to specify the number of records to read since it this test continues to read either type of data until EOT is received. This test is useful for monitoring any BSC line sending data to the processor. For example, the RI test can receive data from the $RJE2780 or $RJE3780 utility operating in the same Series/1 or in another Series/1.

Example: RI command

```
COMMAND (?):  RI
RI ----- READ - TRANSPARENT/NONTRANSPARENT
READ ADDRESS?  5A
READ RECL?  80
READ MONITOR?  Y
```

Testing Write Capability

The WI command tests the write of nontransparent data. The utility prompts you for device address, record length, and number of records.

The WIX command tests the write of transparent data. You specify device address, record length, and number of records.

Example: WI command

```
COMMAND (?):  WI
WI ----- WRITE - NONTRANSPARENT
WRITE ADDRESS?  5B
WRITE RECL?  80
NUMBER OF RECORDS?  10
WRITE MONITOR?  Y
```

Example: WIX command

```
COMMAND (?):  WIX
WIX ---- WRITE - TRANSPARENT
WRITE ADDRESS?  5B
WRITE RECL?  80
NUMBER OF RECORDS?  5
WRITE MONITOR?  Y
```

Ending $BSCUT2 Utility (EN)

The EN command ends the $BSCUT2 utility.

Example: EN command

```
COMMAND (?):  EN
$BSCUT2  ENDED AT 01:14:40
```

Changing Hard-Copy Device (CH)

The CH command reassigns the hard-copy device where the test messages and results are printed or displayed. If the hard-copy device you enter is not defined to the system, output goes to the terminal that loaded $BSCUT2.

Example: CH command

```
COMMAND (?):  CH
NEW HARD-COPY DEVICE?  $SYSLOGA
```

Interpreting Test Results

The results of a test will print out or display at your output terminal. The utility issues a test pattern message for every record it read or wrote in a test.

The first line of a test pattern message gives the task name, record number, and record length.

The second line shows the alphabet repeated to fill up the number of characters specified for record length.

```
TASK READ ENTERED RECORD NUMBER= 1 RECORD LENGTH= 72
ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRST
```

The meanings of the task names are as follows:

- READ - read of standard or transparent data in standard mode
- RXV1 - read of transparent data in conversational mode
- RNV1 - read of standard data in conversational mode
- WRTN - write of standard data in standard mode
- WRIT - write of transparent data in standard mode
- WXV1 - write of transparent data in conversational mode
- WNV1 - write of standard data in conversational mode
- MTX1 - read of transparent data by a tributary station
- MCX1 - write of transparent data by a control station.

The output message in the previous example repeats for the number of records transmitted.

A test can fail if the utility detects an internal error in BSCAM. In that case, the utility issues a BSC return code that points out the cause of the problem. Refer to the *Messages and Codes* for explanations of the BSC return codes. However, a test can also fail because you supplied wrong information to the utility. It could be that you gave invalid BSC line addresses or specified too small a buffer for a read test. Retry the test and specify valid information to the utility.

Whenever an error is detected, either in BSCAM or in the information you entered, the test ends, the utility is cancelled, and a program check message is issued to the logging terminal for the system.

Monitoring BSC Lines with the Communications Indicator Panel

The communications indicator panel is an aid in program debugging and machine troubleshooting during BSCAM operations. It lets you select a BSC line and the activity on function to monitor.

Installing and Attaching the Communications Indicator Panel

The panel can be installed on the frame of full-width processors and I/O expansion units by screwing it into the upper left part of the unit. It is possible to use the panel on half-width processors because it is not necessary to install it on the unit itself.

To monitor a BSC line you must attach the panel connector to the card where the line is assigned. Attach the panel connector to the appropriate set of pins on the card you want to monitor. Be sure the card itself is properly seated and that the connector and pins fit together snugly. Do not force the connector onto the pins.

Selecting BSC Line to Monitor

When using the panel on a 4-line adapter card, you must select which of the BSC lines to monitor. Using the toggle switches labeled "LINE SELECT," set the last three bits of the line's device address, in binary form.

When using the panel on a single-line card, you do not need to set the line select switches. The panel will monitor the line automatically.

Interacting with BSCAM (Using BSC Utilities) *(continued)*

Selecting Function to Monitor

The "DISPLAY/FUNCTION SELECT" switches allow you to select what activity or function to monitor on the line. The settings are different for the single-line adapter and the 4-line adapter. The following is an overview of the functions and activities you can monitor with the panel.

- DCB control word
- DCB chain address
- DCB byte count
- DCB data address
- Storage data register
- Interrupt condition code
- Interrupt status byte
- Cycle-steal status word
- Cyclic redundancy check character
- Modem or modem eliminator states
- Jumper assignments
- Multipoint station address
- Control character transmission
- Errors in block checking.

The information supplied with the communications indicator panel can help you isolate problems or confirm proper functioning of BSCAM and its associated hardware.

Example of Using the Communications Indicator Panel

Figure 22 shows the communications indicator panel. Assume that it is being used to test one of the lines on a 4-line adapter. The "line select" switches are set at "111," indicating the last three digits, in binary form, of the line address being tested. The "display/function select" switches are set at "10111," which causes the panel to monitor when the line goes into select or control mode, and when VRC or BCC errors occur on the line. (For detailed information on the exact switch settings to test both single and 4-line cards, refer to the *IBM Series/1 Communications Features Description*, GA34-0028. For X.21 display/function select switch settings, refer to the *IBM Series/1 Synchronous Communications Single-Line Control Attachment Feature*, GA34-0241.)

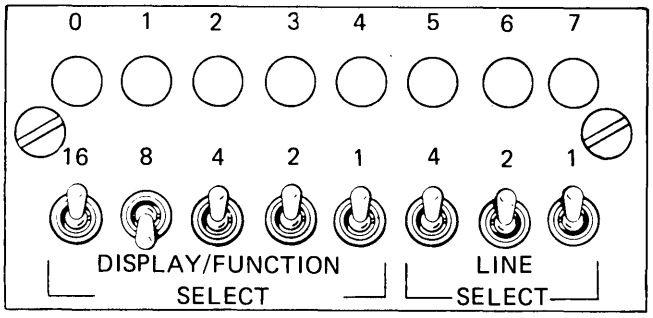


Figure 22. Communications Indicator Panel

Binary Synchronous Communications Access Method (BSCAM)

Using X.21 Switched Network Support

The X.21 circuit switched network support is the basis for the BSC link with the digital Public Data Network used by many countries outside the United States. The information in the preceding sections is still valid for the X.21 network; the exceptions are noted in this section.

For X.21 to function on your Series/1, you must:

- Jumper the IBM 2080 synchronous communications single-line control, high speed feature card for switched operation.
- Perform system generation for X.21 circuit switched support.
  - Define the connection type you want.
- Edit the $$X21DS data set on your IPL volume and build a connection record.
  - Know the network information codes for your country.
- Convert BSC program for X.21.
- Activate $LOG for tracking X.21 errors and call progress signals.

Note: For general hardware information about IBM implementation of X.21, refer to *IBM Implementation of X.21 Interface General Information Manual,* GA27-3287.

Attaching and Jumpering the 2080 Card

You must jumper the 2080 card for switched operation if you want X.21 switched network support capabilities, and you must jumper BSC if you want to use BSC.

For details on the functional characteristics and installation of this card, refer to *IBM Series/1 Synchronous Communication Single-Line Control Attachment Feature Description*, GA34-0241, or the Maintenance Logic Diagrams and the *Customer Site Preparation Manual*, GA34-0050.

System Generation for X.21 Support

During system generation, you must tell the supervisor to support X.21 by defining the BSCLINE definition statement and connection type, and by including the BSCX21 module. (Refer to *Installation and System Generation Guide* for system generation information.)

Using X.21 Switched Network Support *(continued)*

Determining the Connection Type You Need

The following figure shows the four connection types defined for BSC X.21 circuit switched support, their connection method, and their protocol after connection.

BSC connection type supplied by user	X.21 method	BSC protocol after connection
Switched auto (SA)	Auto answer	Pt-to-pt switched
Switched manual (SM)	Auto call	Pt-to-pt switched
Auto call (AC)	Auto call	Pt-to-pt switched
Direct call (DC)	Direct call	Pt-to-pt switched

Figure 23. Mapping Procedures for BSC X.21 Circuit Switched Support

Use the connection type when you code the TYPE= operand of the BSCLINE statement. You must code this parameter because there is *no* default for TYPE= with X.21 circuit switched support. Code this operand at system generation time.

The $$X21DS Connection Record Data Set

You have an IBM-supplied one-record data set allocated on your IPL volume with the reserved name $$X21DS. It contains no information. If you are using TYPE=DC, you can edit this data set to create your own connection records. With TYPE=AC or SM, you can either create your own connection records, or you can create and use the default record named X21RECyy, where "yy" is the hexadecimal address of your attachment card. If you are specifying TYPE=SA (auto answer), the system requires no connection record, but don't delete the $$X21DS data set or X.21 will fail.

Building a Connection Record

You can build as many connection records as you need. To format a connection record, use the $FSEDIT utility or option 1 of the session manager. (For information on the $FSEDIT utility and the session manager, refer to the *Operator Commands and Utilities Reference.*)

Every connection record that you build MUST begin in column 1. Figure 24 shows the format for each connection record, followed by an explanation of each field.

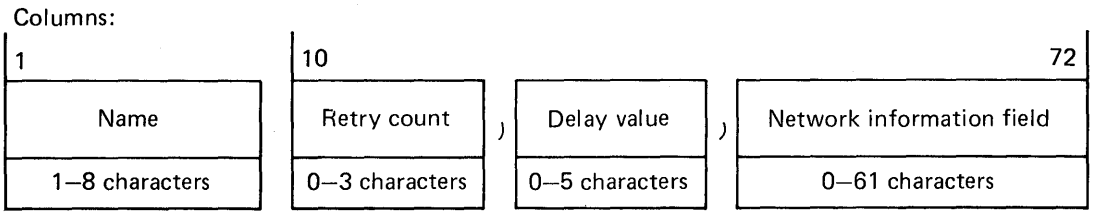


Figure 24. Connection Record Format Fields

- Name - a 1 to 8 alphanumeric character name. The system uses this record name to identify the connection record it should use for a particular request. If you are going to use the default record with auto call (AC or SM) or direct call (DC), one of your record names must be X21RECyy (where yy is the hex address of the 2080 card).

- Retry Count - decimal number from 0 to 255 to indicate the maximum number of times the system should retry this same request. This field MUST begin in column 10. Use a comma to separate this field from the delay value field. If you use a comma by itself, the number of retries defaults to 1.

- Delay Value - decimal number from 0 to 65535 to indicate (in milliseconds) the time between the receipt of an error status from a call and the time when the network should reissue that call. Use a comma to separate this field from the network information field. If you use a comma by itself, the delay time defaults to 0 milliseconds.

- Network Information Field - up to 61 characters for a total record length of 72. This field contains your facility requests and the address selections the network should use for a call request. All information in this field must conform to the requirements of your network, including all special characters. (Refer to the network information technical report for your country to find this information.) If you have specified DC (direct call), the network will not use this field, but it will use the name, retry, and delay fields. Neither the hardware nor the software verifies that the data in the network information field is valid.

The following example shows a sample connection record data set. The fourth sample record, X21RECOA, illustrates the default record. Its retry count is 2 and its delay value is 1. You have to fill in the network information field with valid data for your country. The last sample record, GEORGE, is an example of a connection record for DC (direct call); no network information is needed for DC.

Example: A connection record data set.

```
CONNREC1 255,65000,NETWORK INFORMATION GOES INTO THIS FIELD
CONNREC2 156,100,01234567+
CONNREC3 ,,01234567890+
X21RECOA 2,1,0123456789012345+
GEORGE   25,255,
```

Convert BSC Program for X.21

The only change you need to make for BSC programs to run with X.21 is to code the X21RN operand on the BSCOPEN instruction in your program if you want the system to use your own connection records. (Refer to *Language Reference* for further coding information.)

If you are using auto call (AC or SM) and you don't code X21RN, X.21 will look for the default record named X21RECyy, where yy is the hexadecimal address of your attachment card. As stated earlier, you must insert X21RECyy into the $$X21DS data set. However, if you specify DC and you don't code X21RN, the retry and delay values default to 1 and 0 respectively.

Figure 25 shows a coding example for the BSCOPEN statement. For example, if you name your connection record "CONNREC1," the BSCOPEN statement contains the pointer, X21RN, to that record.

Note: In the DC statement below, if your record name has fewer than 8 characters, it's a good idea to pad the name with blanks to equal 8 characters in length.

```
label     BSCOPEN    BSCIOCB,X21RN=recrdptr
recrdptr  DC         cl8'CONNREC1' member name
```

Figure 25. X.21 BSCOPEN Coding Example

X.21 Error and Call Progress Signal Logging

You should activate $LOG when using X.21 circuit switched support. Then use the LL or LP command of the $DISKUT2 utility to list your error log record. The system will tell you when you have an X.21 switched error. Then you need to check the error log record to determine what the error is.

Figure 26 shows an example of the printed output created by the $DISKUT2 utility when you have X.21 circuit switched support. In this case, the X.21 return code field (word 19) equals -9 (FFF7), indicating a read instruction error. An explanation of the numbered items follows the example.

```
    COMMAND(?): LL
1   LOG DS NAME:  LOGDS

    DEVICE ADDRESS (NULL FOR ALL):  0002
2   I/O LOG ERROR COUNTERS (BY DEVICE ADDR):
                    .
                    .
                    .
                    .
                    .
    REQUESTED HEX DUMP OF LOG RECORD:
              3
    0000      0100 0000 0055 7E00 0000 0222 0000 0000
                                                4 5  6
    0010      0000 0000 0000 0000 0002 02D0 0402 8000
              7    8    9
    0020      FFFF 0005 FFF7 0000 0000 0000 0000 0000
    0030      0000 0000 0000 0000 0000 0000 0000 0000
    0040      0000 0000 0000 0000 0000 0000 0000 0000
    0050      0000 0000 0000 0000 0000 0000 0000 0000
    0060      0000 0000 0000 0000 0000 0000 0000 0000
    0070      0000 0000 0000 0000 0000 0000 0000 0000
    0080      0000 0000 0000 0000 0000 0000 0000 0000
    0090      0000 0000 0000 0000 0000 0000 0000 0000
    00A0      0000 0000 0000 0000 0000 0000 0000 0000
    00B0      0000 0000 0000 0000 0000 0000 0000 0000
    00C0      0000 0000 0000 0000 0000 0000 0000 0000
    00D0      0000 0000 0000 0000 0000 0000 0000 0000
    00E0      0000 0000 0000 0000 0000 0000 0000 0000
    00F0      0000 0000 0000 0000 0000 0000 0000 0000

    LOG LISTING ENDED
```

Figure 26. Example of the X.21 Printed Log Information for a Read Error

1 This is the name of the log data set you created with $LOG. In this example, the log data set is LOGDS.

2 This is what you will see on the usual log information. The dots (.) replace the list of device addresses and I/O error indications that $DISKUT2 provides. These device addresses range from X'00'to X'FF', or 0 to 255.

3 This is the start of your X.21 log record output. The first 28 bytes contain the log header information that is reserved for system use.

4 This byte contains the X.21 record type, X'04'. It marks the beginning of the X.21 statistical log.

5 This byte contains your device address, in this case X'02'.

6 This word contains the X.21 error flags reserved for system use. In this case, it indicates that there are X.21 log entries.

7 This word indicates that there is a read instruction error when equivalent to -1 (FFFF hex). Refer to this byte only when the word indicated by **9** equals -9 (FFF7 hex).

8 This word contains the error return code from the read instruction. Refer to this word only when the word indicated by **9** equals -9 (FFF7 hex). (Refer to *Messages and Codes* for the meanings of these error codes.)

9 When this word equals -9 (FFF7 hex), you must consult the two words indicated by **7** and **8**. In this case, the remainder of the log record will contain zeroes.

Figure 27 shows a second example of the printed output created by $DISKUT2 when you have X.21 circuit switched support. In this case, the X.21 return code field (word 19) equals -27 (FFE5) and indicates a device error. An explanation of the numbered items follows the example.

```
     COMMAND(?): LL
1    LOG DS NAME:  LOGDS

     DEVICE ADDRESS (NULL FOR ALL):  0002
2    I/O LOG ERROR COUNTERS (BY DEVICE ADDR):
                      .
                      .
                      .
                      .
                      .
     REQUESTED HEX DUMP OF LOG RECORD:
               3
     0000      0100 0000 0055 7E00 0000 0222 0000 0000
                                                4 5  6
     0010      0000 0000 0000 0000 0002 02D0 0402 C000
                         7    8 9
     0020      0000 0000 FFE5 03FA 0000 0000 0000 0000
                                                   10
     0030      0000 0000 0000 0000 0000 0000 0002 0000
     0040      0000 0000 0000 0000 0000 0000 0000 0000
     0050      0000 0000 0000 0000 0000 0000 0000 0000
                         11
     0060      0000 0000 0001 0000 0000 0000 0000 0000
     0070      0000 0000 0000 0000 0000 0000 0000 0000
                                          12
     0080      0000 0000 0000 0000 0000 0000 0000 0000
     0090      0000 0000 0000 0000 0000 0000 0000 0000
     00A0      0000 0000 0000 0000 0000 0000 0000 0000
     00B0      0000 0000 0000 0000 0000 0000 0000 0000
     00C0      0000 0000 0000 0000 0000 0000 0000 0000
     00D0      0000 0000 0000 0000 0000 0000 0000 0000
     00E0      0000 0000 0000 0000 0000 0000 0000 0000
     00F0      0000 0000 0000 0000 0000 0000 0000 0000

     LOG LISTING ENDED
```

Figure 27. Example of the X.21 Printed Log Information for a Device Error

1 This is the name of the log data set you created with $LOG. In this example, the log data set is LOGDS.

2 This is what you will see on the usual log information. The dots (.) replace the list of device addresses and I/O error indications that $DISKUT2 provides. These device addresses range from X'00'to X'FF', or 0 to 255.

3 This is the start of your X.21 log record output. The first 28 bytes contain the log header information that is reserved for system use.

4 This byte contains the X.21 record type, X'04'. It marks the beginning of the X.21 statistical log.

5 This byte contains your device address, in this case X'02'.

6 This word contains the X.21 error flags reserved for system use. In this case, it indicates that there are X.21 log entries, and that a device error has occurred.

7 This word is the X.21 return code field. When it equals -27 (FFE5 hex), consult the device error code field (**9**).

8 This byte shows you how many times the call was retried before it failed. In this case, the retry count equals 3.

9 This byte will give you the device error code, in this case -6 (FA hex). Use the data in this byte only when the word indicated by **7** equals -27. The error codes are as follows:

```
DEVICE ERROR CODES
 -1 (FF)     Buffer overrun
 -2 (FE)     Unsuccessful DCE clear
 -3 (FD)     Interface data check error
 -4 (FC)     Invalid interrupt code
 -5 (FB)     Invalid interrupt status byte
 -6 (FA)     Invalid I/O condition code
 -7 (F9)     Start cycle steal status issued
 -8 (F8)     Specification check
```

Figure 28. Device Error Codes

Note: Refer to the hardware manual *IBM Series/1 Communications Theory Diagrams*, SY34-0059, for the meanings of these messages.

The next 100 bytes (00 to 99) are the call progress signal counters. They record call progress signals and hardware errors. The following example shows the meaning of the significant bytes. The number within the byte indicates how many times the error occurred. For example, 10 shows you a call progress signal of 21 because it's the 21st. byte after 9; the number 02 within the byte tells you that it occurred twice. 11 shows you a call progress signal 61 because it's the 61st. byte after 9; the number 01 within the byte tells you that it occurred once. The byte number column gives the byte number relative to the beginning of the log (3).

Call progress signal	Byte number	Meaning of signal	What X.21 does
00 01 02 03	28 29 2A 2B	Reserved Terminal called Redirected call Connect when free	Does not clear. Waits for attempt to complete.
20 21 22 23	3C 3D 3E 3F	No connection Number busy Selection signals procedure error Selection signal transmission error	Clears due to short-term conditions. Tries again up to retry limit.
41 42 43 44 45 46 47 48 49 51 52	51 52 53 54 55 56 57 58 59 5B 5C	Access barred Changed number Not obtainable Out of order Controlled not-ready Uncontrolled not-ready DCE power off Invalid facility request Network fault in local loop Call information service Incompatible user class of service	Clears due to long-term conditions. Call unsuccessfully completed.
61	65	Network congestion	Clears due to network short-term conditions. Tries again up to retry limit.
71 72	6F 70	Long-term network congestion RPOA out of order	Clears due to long-term network conditions. Call unsuccessfully completed.
81 82 83	79 7A 7B	Registration/cancellation confirmed Redirection activated Redirection deactivated	Clears due to DTE network procedure.

Figure 29. Call Progress Signal Counter Usage

12 This byte marks the end of the statistical log.

Chapter 2. Remote Management Utility ($RMU)

When the Remote Management Utility ($RMU) is loaded on a Series/1, it allows another system, called the host, to control the Series/1. The Series/1 with $RMU loaded on it is called the remote system.

The host starts and controls functions that $RMU performs on the remote system. $RMU waits for an application program running on the host to ask it to perform some function, and then does the work. No operator action at the remote system is needed. $RMU sends responses to the host that tell if it completed the function successfully, and to provide other information about the function.

The $RMU-controlled (remote) system is always a Series/1. The host system can be a Series/1, too. This chapter talks about the host system being a Series/1. It tells how to write host programs to communicate with $RMU. The Binary Synchronous Communications Access Method (BSCAM) controls I/O during $RMU operations. Because of this, $RMU operation requires the use of binary synchronous communication (BSC) lines. In addition, the host program must consist of Event Driven Language BSC instructions and must follow rules for BSCAM programs in general.

If your Series/1 is the Host system making requests of another Series/1, then you must write the application programs that send requests to $RMU.

Your host program can ask $RMU to perform the following functions:

- Manage data sets on the remote system
  - ALLOCATE function
  - DELETE function
  - DUMP function.
- Transfer data between the two systems
  - SEND function
  - RECEIVE function
  - WRAP function.
- Control the running of programs on the remote system
  - EXEC function
  - SHUTDOWN function.
- Establish interactive sessions between the two systems
  - PASSTHRU function.
- Verify the IDs of the two systems
  - IDCHECK function.

If your Series/1 is the remote system, there is no work that you must do to respond to host requests, since the utility takes care of that automatically. However, before $RMU operations can begin, you must load the utility into your system with the command **$L $RMU**. The only other work that can be done at the remote system is changing $RMU default values. The section, "Remote Management Utility Defaults" on page CO-62 tells about these values and how to change them.

Figure 30 on page CO-59 shows how the host and remote systems communicate by means of the host program and $RMU.

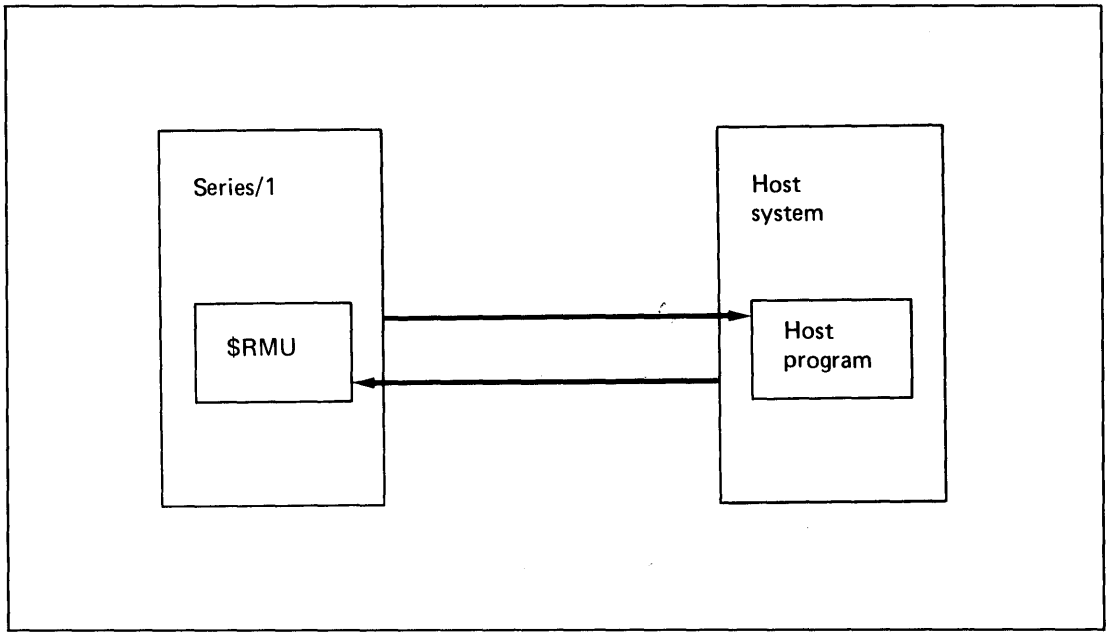


Figure 30. Communication Between Host and Remote Systems

Planning for the Remote Management Utility Operations

Types of Line Connections

$RMU operations can take place over several types of BSC line connections.

The host and remote systems can be connected on either point-to point or multipoint lines during $RMU operations. Point-to-point connections can be over leased or switched lines, depending on which type of service you have bought from the common carrier. Multipoint connections require the host system to be the control station and the remote system to be a tributary station. Remember, $RMU operates in point-to-point only for X.21.

Find out what type of line connection your Series/1 is part of, and whether it is acting as the host or the remote system. Keep this information in mind since you will use it in other areas of planning for $RMU operations.

Planning for the Remote Management Utility Operations *(continued)*

Mode of Transmission

Transmissions during $RMU operations are in transparent mode. Although BSCAM, which controls I/O for $RMU, supports other modes of transmission, only the transparent mode is available with $RMU. This is important when you write host programs to send requests to $RMU.

Storage Considerations

$RMU needs a maximum of 7.25K bytes of storage, plus buffer space, to perform all its functions. However, $RMU can perform its data set management functions with only 5.5K bytes of storage. In that case, $RMU gets any additional storage it needs from the partition in which it is executing.

The section, "Remote Management Utility Defaults" on page CO-62 tells how to use the reduced storage size and how to change it.

Remote System Requirements

For $RMU operations the remote Series/1 must meet certain hardware and software requirements.

Hardware Requirements

Minimum hardware requirements for the remote Series/1 are as follows:

- *Processor*: 4952, 4954, 4955, or 4956.
- *Storage*: must be sufficient to handle the supervisor, $RMU (maximum of 7.25K bytes), and the programs that run under $RMU (see notes).
- *BSC Hardware Connection Features*: must be one of the following:

  single-line control, medium speed

  single-line control, high speed

  8-line control and one or two 4-line adapters

  multifunction attachment.

- *Disk or Diskette*: 4962, 4963, or 4967 (disk); 4964, 4965, or 4966 (diskette).

Notes:

1. The section "Storage Considerations" on page CO-60 tells you how $RMU uses storage.

2. For additional information on storage considerations, refer to the *Installation and System Generation Guide*.

Software Requirements

Since the Binary Synchronous Communications Access Method (BSCAM) controls the transfer of data during $RMU operations, the system generation for the remote Series/1 must include one BSCLINE statement per copy of $RMU you plan to use. You must define each BSC line to be used by $RMU as either a point-to-point (BSCLINE TYPE=PT, SM or SA), or multipoint tributary station (BSCLINE TYPE=MT). You must include enough storage to accommodate $RMU and the programs that run under it in the partition where you plan to load it.

If you plan to set up PASSTHRU sessions between the remote Series/1 and the host, $RMU requires two virtual terminals. You must include two TERMINAL statements in your system generation. On one statement, code the parameter ADDRESS=CDRVTA, and on the other, code ADDRESS=CDRVTB. These are the only addresses that are valid.

You must also include support for the BSCAM supervisor module.

Refer to the *Installation and System Generation Guide* for details on performing system generation.

Host System Requirements

In this discussion, the host system is a Series/1. It too must meet certain requirements to successfully communicate with $RMU. The following support must be defined during system generation for the host Series/1:

- BSCLINE TYPE=PT, SM, SA, or MC
- BSCAM supervisor module support.

Besides the Series/1, the host can be any system that meets these requirements:

- provides binary synchronous protocol compatible with BSCAM
- transmits in transparent EBCDIC
- supports the form of record exchange that $RMU performs.

Remote Management Utility Defaults

Certain values associated with $RMU operations have predefined defaults. These default values are:

- Host system ID is set at HOSTRMUX.
- Remote system ID is set at REMTRMUX.
- BSC line is set at X'09'.
- $RMU storage size is set at 7.25 K bytes.
- Buffer size is set at 1024 bytes.

At the remote system, you can modify these default values by using the $DISKUT2 utility. This utility allows you to get at main storage for $RMU and patch your changes.

After you load the $DISKUT2 utility, respond to its prompts to make changes to the default values. In general, the utility prompts for the following information:

- a $DISKUT2 command
- the storage address containing the default value
- type of code the default value is in
- new data to replace the default value.

The storage addresses listed in this book are subject to change. Consult the PID directory for the latest storage addresses for $RMU.

The sections that follow tell what information to enter to change each of the defaults.

Changing Host System ID

The default value for the host system ID (used in the IDCHECK function) is HOSTRMUX. You can change this to another name if you wish. In response to the utility's prompts, enter the following information:

- Command: PA $RMU
- Address: 0864 (*)
- Code type: E (for EBCDIC)
- Data: the new host system ID (8 characters).

* The address listed for the host system ID is subject to change. Consult the PID directory for the latest storage addresses for this default value.

Changing the Remote System ID

The default value for the remote system ID (used in IDCHECK function) is REMTRMUX. You can change this to another name if you wish. In response to the utility's prompts, enter the following information:

- Command: PA $RMU
- Address: 085C (*)
- Code type: E (for EBCDIC)
- Data: the new remote system ID (8 characters).

* The address listed for the remote system ID is subject to change. Consult the PID directory for the latest storage addresses for this default value.

Planning for the Remote Management Utility Operations *(continued)*

Changing the BSC Line Address

The default value for the BSC line address is X'09'. You can change this to another line address if you wish, but if you do, make sure that the BSC line definitions for the remote system (made during system generation) reflect the change. In response to the utility's prompts, enter the following information:

- Command: PA $RMU
- Address: 086E (*)
- Code type: H (for hexadecimal)
- Data: new BSC line address.

* The address listed for the BSC line address is subject to change. Consult the PID directory for the latest storage addresses for this default value.

Changing Storage Size

The default value for storage required by $RMU is 7.25K bytes. However, you can change the size of storage from 7.25 to 5.5K bytes. In response to the utility's prompts, enter the following information:

- Command: SS $RMU
- New Storage Size in Bytes: nnnn, which specifies the new storage size.

Changing Buffer Size

The default value for buffer size on the remote system is 1024 bytes. You can change buffer size within the minimum value of 512 and the maximum value of 32,512 bytes. Buffer size should be in multiples of 256. In addition, buffer size depends on the size of the blocks $RMU is storing in the buffer. Two factors determine the size you should change buffer size to:

- data set type to be stored in the buffer
- number of blocks sent in each record, according to data set type.

First decide what data set type you will be storing in the buffer.

- Standard data set - meant to contain standard data
- Source data set - meant to contain source data
- PASSTHRU data set - meant to be used in a PASSTHRU session.

Now determine the blocking factor for your type of data set. The blocking factor refers to the number of logical records contained in each block of data. If you want to send a certain number of records in each block, then your buffer must be able to accommodate those records. The calculations to determine blocking factor are described in the sections that follow.

Determining Blocking Factor for Standard Data Sets: To determine the blocking factor for a standard data set, make the following calculation:

(buffer size - 6) / 256 = blocking factor

Discard the remainder.

For example, assume a buffer size of 768 (256 x 3); subtract 6 to get the value 762. Divide 762 by 265; the quotient is 2. Discard the remainder. The blocking factor is 2.

Planning for the Remote Management Utility Operations *(continued)*

Determining the Blocking Factor for Source Data Sets: To determine the blocking factor for a source data set, make the following calculation:

(buffer size - 262) / 80 = blocking factor

Discard the remainder.

For example, assume a buffer size of 2048 (256 x 8); subtract 262 to get the value 1786. Divide 1786 by 80; the quotient is 22. Discard the remainder. The blocking factor is 22.

Determining Blocking Factor for PASSTHRU Data Sets: To determine the blocking factor for a PASSTHRU data set, make the following calculation:

buffer size - 264 = blocking factor
Discard the remainder.

For example, assume a buffer size of 512 (256 x 2); subtract 264. The result is 248. The blocking factor is also 248.

To change the buffer size, enter the following information in response to the utility's prompts:

- Command: PA $RMU
- Address: 0892 (*)
- Data: nnnn, which specifies the new buffer size.

* The address listed for the buffer size is subject to change. Consult the PID directory for the latest storage addresses for this default value.

Host Programming for the $RMU Application

If your Series/1 is acting as the host, you must write application programs to communicate with $RMU on the remote Series/1. Your program sends requests to $RMU and receives responses from $RMU.

Using Event Driven Language BSC Instructions

Since BSCAM controls transmissions between the host and remote systems, your host program must contain EDL BSC instructions. To send requests, code BSCWRITE instructions, and to receive $RMU's responses, code BSCREAD instructions.

You may want to review BSCAM programming techniques before going further with programming for the $RMU application. Chapter 1 contains information on BSCAM programming using the BSC instructions.

Since $RMU transmissions are in transparent mode, the BSC write instructions in your program must be of the transparent type. For example, to send a request, code a BSCWRITE IX (initial transparent write). To signify the end of a request, code a BSCWRITE E instruction. To send data, code BSCWRITE IX for the first record and BSCWRITE CX (continue transparent write) for subsequent records.

To review the syntax of the BSC instructions, refer to the *Language Reference.*

Receiving $RMU's Responses to Host Requests

During the course of performing any function, $RMU sends various types of messages to the host. These messages provide the following information to the host:

- Status messages that indicate success or failure of a function
- Count messages that show the number of records sent or received by $RMU
- Data messages that $RMU uses to send data.

All of these messages start with three fields that make up the header, as shown below:

- RMHBSCC contains the BSC control characters DLE STX.
- RMHID identifies a message from $RMU.
- RMHTYP identifies the type of message: 'S' for status, 'C' for count, or 'D' for data.

Status Message

$RMU sends a status message to the host to indicate the success or failure of a requested function. A status record is 18 bytes in length. Besides the three fields that make up the header, it contains several fields that provide the following information:

- RMSREQ specifies the request that the status message pertains to. For example, if it is in response to an allocate request, the RMSREQ field shows the value 2, which represents the allocate request.
- RMSFN indicates the success or failure of the particular request. This field contains a -1 to indicate the success of a request. Any other value (a positive value) indicates failure of the request. The number refers to a specific error condition, as shown in the chart below:

Status Code	Condition
1	IDCHECK Function Failed
2	Buffer area is too small for the record
3	Short record (less than 4 bytes)
4	Header ID is 'H' (invalid)
5	Invalid header ID (not 'X' or 'H')
6	Request expected
7	Invalid request
8	Request short (missing information)
9	Invalid SEND/RECEIVE type
10	Invalid blocking factor
11	Invalid message received during request
12	Invalid PASSTHRU record type
13	Invalid DUMP partition number
14	Request received while another was running
15	EOT expected and not received
16	Virtual terminal busy
21	READ disk/diskette failed
22	WRITE disk/diskette
24	Load failed
25	Load of overlay failed
26	BSC I/O failure
27	PRINTEXT failed for virtual terminal
31	ALLOCATE/DELETE failed
32	OPEN failed
33	SETEOD failed
34	Parameters to build LOAD instructions are invalid
41	Overlay function missing

Figure 31. $RMU Status Failure Codes

- RMSST appears when the number in RMSFN indicates that an Event Driven Executive function failed. This field appears in a status message only if one of the following numbers appears in the RMSFN field:

  - 21 or 22: contains disk (READ/WRITE) return code
  - 24 or 25: contains LOAD return code
  - 26: contains BSC (binary synchronous communications) return code
  - 27: contains virtual terminal I/O return code
  - 31, 32 or 33: contains $DISKUT3 return code
  - 34: contains LOAD return code.

  For explanations of the return codes, refer to the *Messages and Codes*.

- RMSRID appears only in the status message of a successful IDCHECK request. It specifies the ID of the remote Series/1.

Count Message

$RMU sends a count message to the host when it detects an end-of-data condition during a data set transfer (from either a SEND or RECEIVE request). This message shows the number of records that $RMU sent. The count message also indicates if records were padded (blanks inserted) during the data set transfer. The host should use the count message to verify whether a complete data transfer occurred. For example, if the host program sent 30 messages, then the count message from $RMU should indicate that the utility received 30 messages.

The count message can be up to 12 bytes long. Besides the header, it contains several fields that provide the following information:

- RMCREQ identifies the request type that the count message pertains to (0 = SEND, 1 = RECEIVE).
- RMCFLG indicates if record padding occurred during a data set transfer. A value of '1' in this field indicates padding; a value of '0' indicates no padding.
- RMCCNT specifies the number of records transmitted. This number reflects the number of logical records (80-byte or 256-byte) that $RMU transmitted, regardless of how the records were blocked.

Data Message

$RMU sends data messages to transmit data to the host in response to a SEND request. This type of message contains the 80-byte or 256-byte records from the data set specified in the SEND request.

Besides the header, the data record contains the field RMDDATA. RMDDATA contains the data that $RMU is sending to the host. The length of this field will be a multiple of 80 or 256, depending on the specifications of the host in its SEND request.

Error Handling During $RMU Operations

If a communications error occurs while $RMU is executing, the terminal that loaded $RMU (on the remote system) receives an error message. If a communications error occurs while $RMU is performing a function, it generally terminates. However, the SEND, RECEIVE, and PASSTHRU functions may continue executing because these functions require multiple message exchanges between the host and the Series/1 before the function is complete. If the error is recoverable, $RMU sends the host a status record followed by a termination (EOT). After this sequence of messages is over, the host can issue a new request.

Host Programming for the $RMU Application *(continued)*

Both $RMU and the host program can detect errors while an $RMU function is executing. If $RMU detects such an error, it sends the host a status record indicating the error condition, followed by an EOT to terminate the function. After this sequence is complete the host can issue a new request. If the host program detects an error, it should terminate the function in the same sequence as $RMU. However, the status record the host sends to the remote system requires only the 4-byte header of a status record (RMHBSCC, RMHID, and RMHTYP fields).

$RMU detects errors during all phases of operations and sends failure status messages to the host. Refer to "Status Message" on page CO-67 for details of these failure messages. Status, count and data messages were discussed previously in "Receiving $RMU's Responses to Host Requests" on page CO-67. The type and sequence of responses $RMU sends varies according to the request type. The sections below, which tell how to code each type of request in a host program, also show the way $RMU responds to each request.

You must code a BSCREAD instruction to receive each of $RMU's responses to a request. To receive $RMU's first response to a request, code a BSCREAD I (initial read) instruction. To receive the rest of $RMU's responses, code BSCREAD C (continue read) instructions.

Coding the Required Field for Requests to $RMU

For each request the host sends to $RMU, you must code certain fields of information in your program. Each set of fields identifies a different type of request, and tells $RMU exactly what the host wants it to do.

The sections that follow identify the required fields for each type of request and show what information to enter in each field.

Managing Disk/Diskette Data Sets

The host can ask $RMU to work with disk or diskette data sets on the remote system. The three such functions that $RMU can perform are:

- ALLOCATE a disk or diskette data set
- DELETE a disk or diskette data set
- DUMP storage to a disk or diskette data set.

Allocating Disk/Diskette Data Sets (ALLOCATE)

The host can ask $RMU to allocate a disk or diskette data set on the remote system with the ALLOCATE request. The data set can contain either standard data or a program.

To send $RMU your ALLOCATE request, code a BSCWRITE IX instruction along with the required information fields. Then code a BSCWRITE E or EX instruction to signify that you are finished sending the ALLOCATE request.

After $RMU gets the host's ALLOCATE request, it sends a status message. Code a BSCREAD I instruction to receive this message. After performing the requested function, or after encountering a failure condition, $RMU terminates the function by sending an EOT (end-of-transmission) sequence to the host. Code a BSCREAD C instruction to receive the notice of termination.

IMPORTANT: The ALLOCATE function uses the $DISKUT3 utility. Do not allocate a data set named $EDXNUC, $$EDXVOL, or $$EDXLIB.

Allocating a Program Data Set: If you are going to allocate a data set that is to contain a program from the host system, you must have the following information available:

- The load address of the program
- the size (in bytes) of the program
- the entry point of the program
- the RLD (relocation dictionary) count of the program.

Obtain the information by opening the host program data set and examining the data set control block.

Coding the Required Fields for ALLOCATE

The chart that follows shows all the fields to code in your program for an allocate request. The fields identified with an asterisk (*) contain variable values. You can code any appropriate value in these fields. However, the other fields can contain only the values shown in the chart. Code these fields exactly as the chart specifies.

Field	Size, Type	Explanation	What to Code
RMHBSCC	2 hex	Starts transmission of a request	RMHBSCC DATA X'1002'
RMHID	1 alpha	Identifies message to $RMU	RMHID DATA C'X'
RMHTYP	1 alpha	Identifies request to $RMU	RMHTYP DATA C'R'
RMREQ	2 num	Specifies ALLOCATE request	RMREQ DATA F'2'
RMADSN (*)	8 alpha	Name of the data to be allocated	RMADSN DATA CL8'DATASET'
RMAVOL (*)	6 alpha	Name of the volume containing data set. Default = IPL volume.	RMAVOL DATA CL6'VOLUME'
RMANREC (*)	4 num	Number of 256-byte records allocated for data set	RMANREC DATA D'nn'
RMADST (*)	2 num	Type of data set allocated (1 = data; 3 = program)	RMADST DATA F'n'
RMALAD (*)	2 num	Load address of program data set (see note)	RMALAD DATA F'nn'

Figure 32 (Part 1 of 2). Required Fields for ALLOCATE Request

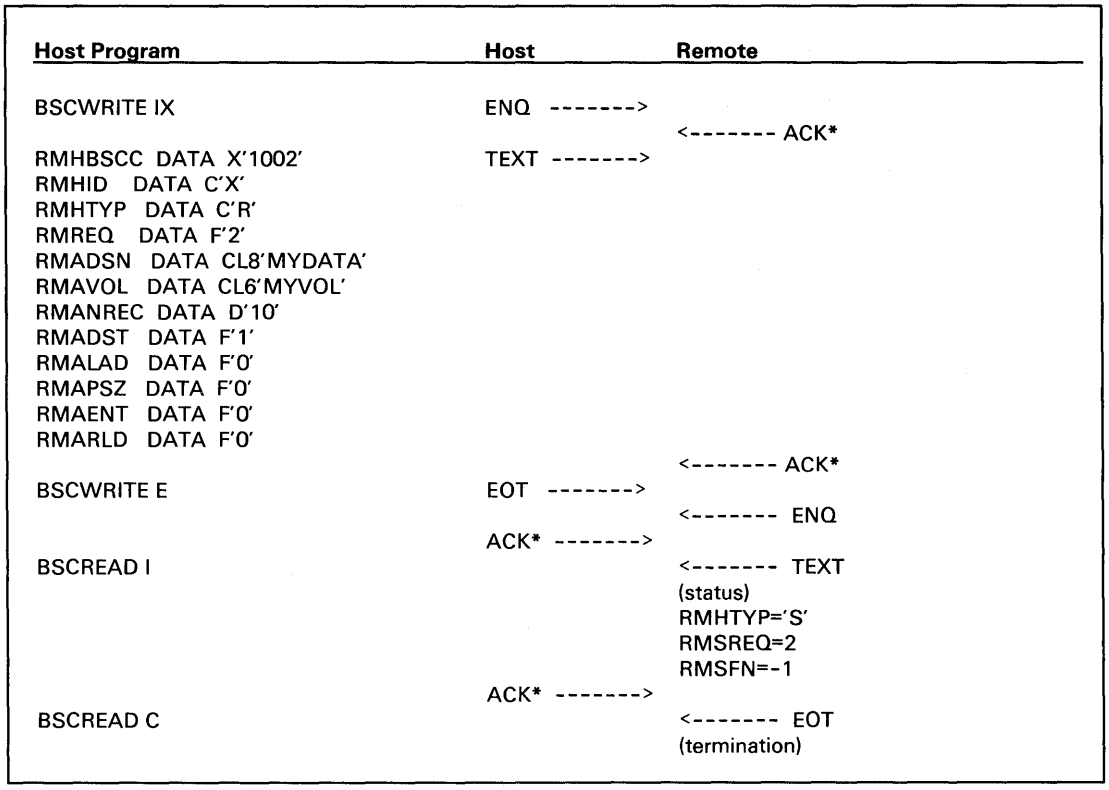
Field	Size, Type	Explanation	What to Code
RMAPSZ (*)	2 num	Program size in bytes (see note)	RMAPSZ DATA F'nn'
RMAENT (*)	2 num	Program entry point (see note)	RMAENT DATA F'nn'
RMARLD (*)	2 num	RLD count of program (see note)	RMARLD DATA F'nn'

Figure 32 (Part 2 of 2). Required Fields for ALLOCATE Request

Note: The RMALAD, RMAPSZ, RMAENT and RMARLD fields are required only if the data set you are allocating is a program. In this case, the RMASDT field must contain the value 3.

Figure 33 shows an example of the ALLOCATE function. The host requests a data set named "MYDATA" to be allocated on volume "MYVOL." The data set type is 1 (data) and ten 256-byte records are allocated. $RMU sends a status record with -1 (successful completion) to the host.

Figure 53 on page CO-109 shows a sample program that can send an ALLOCATE request.



Host Program	Host	Remote
BSCWRITE IX	ENQ ------->	
		<------- ACK*
RMHBSCC DATA X'1002'	TEXT ------->	
RMHID DATA C'X'		
RMHTYP DATA C'R'		
RMREQ DATA F'2'		
RMADSN DATA CL8'MYDATA'		
RMAVOL DATA CL6'MYVOL'		
RMANREC DATA D'10'		
RMADST DATA F'1'		
RMALAD DATA F'0'		
RMAPSZ DATA F'0'		
RMAENT DATA F'0'		
RMARLD DATA F'0'		
		<------- ACK*
BSCWRITE E	EOT ------->	
		<------- ENQ
	ACK* ------->	
BSCREAD I		<------- TEXT
		(status)
		RMHTYP='S'
		RMSREQ=2
		RMSFN=-1
	ACK* ------->	
BSCREAD C		<------- EOT
		(termination)

Figure 33. Communications Flow for ALLOCATE

Deleting a Disk/Diskette Data Set (DELETE)

You can ask $RMU to delete a disk or diskette data set on the remote system with a DELETE request.

To send $RMU your DELETE request, code a BSCWRITE IX instruction along with the required information fields. Then code a BSCWRITE E or EX instruction to signify that you are finished sending the DELETE request.

After $RMU gets the host's DELETE request, it sends a status message. Code a BSCREAD I instruction to receive this message. After performing the requested function, or after encountering a failure condition, $RMU terminates the function by sending an EOT (end-of-transmission) sequence to the host. Code a BSCREAD C instruction to receive the notice of termination.

IMPORTANT: The DELETE function uses the $DISKUT3 utility. Do not delete data sets $EDXNUC, $$EDXVOL, and $$EDXLIB.

Coding the Required Fields for DELETE

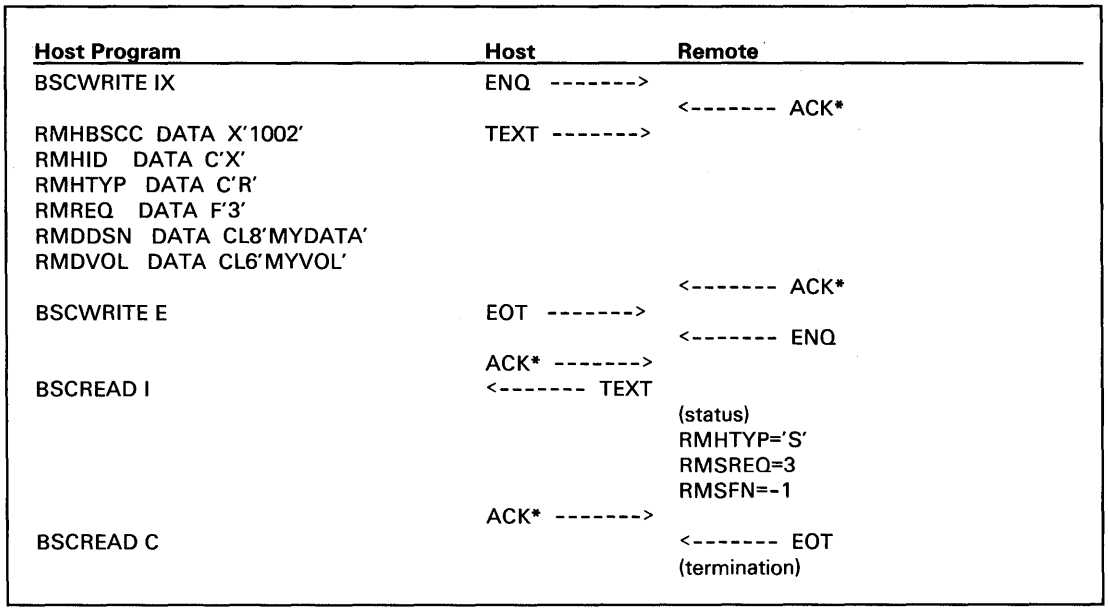
The chart that follows shows all the fields to code in your program for a delete request. The fields identified with an asterisk (*) contain variable values. You can code any appropriate value in these fields. However, the other fields can contain only the values shown in the chart. Code these fields exactly as the chart specifies.

Field	Size, Type	Explanation	What to Code
RMHBSCC	2 hex	Start of message	RMHBSCC DATA X'1002'
RMHID	1 alpha	Identifies message to $RMU	RMHID DATA C'X'
RMHTYP	1 alpha	Identifies request to $RMU	RMTYP DATA C'R'
RMREQ	2 num	Specifies DELETE function	RMREQ DATA F'3'
RMDDSN (*)	8 alpha	Data set to be deleted	RMDDSN DATA CL8'DATASET'
RMDVOL (*)	6 alpha	Volume containing data set to be deleted. Default = IPL volume	RMDVOL DATA CL6'VOLUME'

Figure 34. Required Fields for DELETE Request

Figure 35 on page CO-76 shows an example of the DELETE function. The host specifies a data set named "MYDATA" to be deleted from volume "MYVOL." $RMU sends a status record with -1 (successful completion) to the host.

Figure 53 on page CO-109 shows a program that can send a DELETE request.



Host Program	Host	Remote
BSCWRITE IX	ENQ ------->	
		<------- ACK*
RMHBSCC DATA X'1002'	TEXT ------->	
RMHID DATA C'X'		
RMHTYP DATA C'R'		
RMREQ DATA F'3'		
RMDDSN DATA CL8'MYDATA'		
RMDVOL DATA CL6'MYVOL'		
		<------- ACK*
BSCWRITE E	EOT ------->	
		<------- ENQ
	ACK* ------->	
BSCREAD I	<------- TEXT	
		(status)
		RMHTYP='S'
		RMSREQ=3
		RMSFN=-1
	ACK* ------->	
BSCREAD C		<------- EOT
		(termination)

Figure 35. Communications Flow for DELETE

Dumping Storage to a Disk/Diskette Data Set (DUMP)

You can ask $RMU to dump a storage partition to a disk or diskette data set on the remote system with a DUMP request.

To send $RMU your DUMP request, code a BSCWRITE IX instruction along with the required information fields. Then code a BSCWRITE E or EX instruction to signify that you are finished sending the DUMP request.

After $RMU gets the host's DUMP request, it sends a status message. Code a BSCREAD I instruction to receive this message. After performing the requested function, or after encountering a failure condition, $RMU terminates the function by sending an EOT (end-of-transmission) sequence to the host. Code a BSCREAD C instruction to receive the notice of termination.

Coding the Required Fields for DUMP

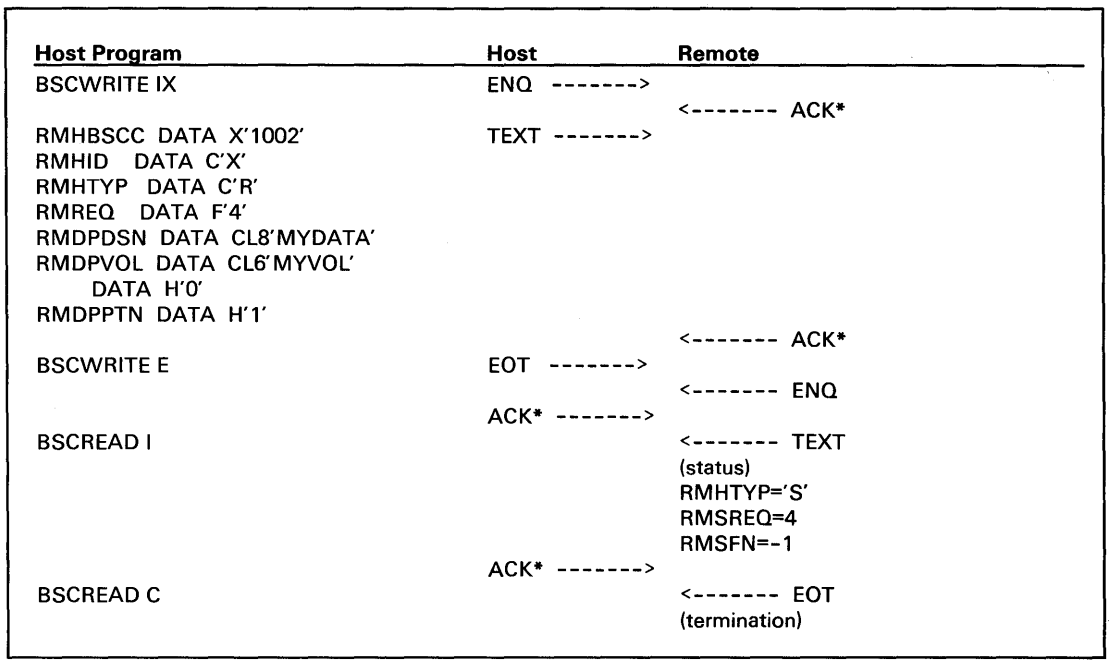
The chart that follows shows all the fields to code in your program for a dump request. The fields identified with an asterisk (*) contain variable values. You can code any appropriate value in these fields. However, the other fields can contain only the values shown in the chart. Code these fields exactly as the chart specifies.

Field	Size, Type	Explanation	What to Code
RMHBSCC	2 hex	Start of transmission	RMHBSCC DATA X'1002'
RMHID	1 alpha	Identifies message to $RMU	RMHID DATA C'X'
RMHTYP	1 alpha	Identifies request to $RMU	RMHTYP DATA C'R'
RMREQ	2 num	Specifies DUMP request	RMREQ DATA F'4'
RMDPDSN (*)	8 alpha	Name of the data set to dump to	RMDPDSN DATA CL8'DATASET'
RMDPVOL (*)	6 alpha	Volume containing dump data set. Default = IPL volume	RMDPVOL DATA CL6'VOLUME'
(filler)	1 n/a	Reserved field (unused)	DATA H'0'
RMDPPTN (*)	1 num	Partition to be dumped	RMDPPTN DATA H'n'

Figure 36. Required Fields for DUMP Request

Figure 37 on page CO-78 shows an example of the DUMP request. The host requests that partition 1 be dumped to the data set "MYDATA" on volume "MYVOL." $RMU sends a status record with -1 (successful completion) to the host.

Figure 53 on page CO-109 shows a sample program that can send a DUMP request.



Host Program	Host	Remote
BSCWRITE IX	ENQ ------->	
		<------- ACK*
RMHBSCC DATA X'1002' RMHID DATA C'X' RMHTYP DATA C'R' RMREQ DATA F'4' RMDPDSN DATA CL8'MYDATA' RMDPVOL DATA CL6'MYVOL' DATA H'0' RMDPPTN DATA H'1'	TEXT ------->	
		<------- ACK*
BSCWRITE E	EOT ------->	
		<------- ENQ
	ACK* ------->	
BSCREAD I		<------- TEXT (status) RMHTYP='S' RMSREQ=4 RMSFN=-1
	ACK* ------->	
BSCREAD C		<------- EOT (termination)

Figure 37. Communications Flow for DUMP

Controlling Data Transfers between Host and Remote Systems

Your host program can ask $RMU to perform functions involving data transfers. You can ask $RMU to send data to your system or receive data from you. Also, you can send data to the remote system and have $RMU echo it back to you.

The requests associated with these functions are:

- RECEIVE data from the host
- SEND data to the host
- WRAP host data back to the host.

Host Sending Data to Remote System (RECEIVE)

You can ask $RMU to receive a host data set, and then put the data into a data set on the remote system. This is the RECEIVE request.

The RECEIVE function requires a data set on the remote system in which to place the data. If such a data set does not already exist on the remote system, you can allocate one with an ALLOCATE request.

Send your RECEIVE request by coding a BSCWRITE IX instruction, along with the required information fields.

Upon receiving the RECEIVE request, $RMU checks to see if it can handle the size of the records to be sent. It then sends a status message to the host. A status message of -1 (successful completion) indicates the RECEIVE function will continue; otherwise it terminates.

After receiving a status message of -1, the host begins to send the data set to $RMU. The record length of the data set should be a multiple of 256 or 80.

If $RMU receives a data record whose length is not a multiple of the specified length, it pads the record with null bytes. For example, a record of 156 bytes will contain padding with 100 null bytes, if 256 bytes is the specified length.

If $RMU receives a data record whose length is greater than the length specified on the request, the RECEIVE function terminates with a status indicating "BSC I/O Failure" and BSC return code 20 (wrong length record - long).

At the completion of the data set transfer, $RMU sends the host a count message to report the number of host records it received. This message also indicates if any records were padded.

If the host is sending an empty data set, send one data record which contains no data (only the 4-byte header) and then end the transmission.

If an unrecoverable error occurs, such as a disk or diskette error, $RMU interrupts the host transmission by sending an EOT (end-of-transmission) and a status message containing the appropriate error code. $RMU terminates the RECEIVE function, and then waits for another request from the host. The host should use the status record to determine the reason for failure.

The host can terminate the RECEIVE function at any time by sending a status message followed by a BSCWRITE E instruction.

Specifying Data Set Type

You must specify what type of data set the host is sending. The field RMRTYP is where to code this information. Enter one of the following values in this field:

- 0 for a standard data set with 256-byte records
- 1 for a source data set with 80-byte records.

Specifying Record Blocking

You must specify whether or not the host will send blocked records to $RMU. If the host is going to send blocked records, you must specify the number of blocks in each record.

The RMRBLK field is where to code this information. Enter one of the following values in this field:

- '0' or '1' to specify no blocking
- any other number to specify the exact number of blocks the host will send in a record.

Specifying the Starting Record

You must tell $RMU which record of the host data set will be the first to be received. For example, you may want to start at the first record, or at any other record within the data set.

The field RMRSTR is where to code this information. If you enter the value '0' or '1', the host will start sending at the first record. If you enter any other value, the host will start sending at that particular record.

Coding the Required Fields for RECEIVE Request

The chart that follows shows all the fields to code in your program for a RECEIVE request. The fields identified by an asterisk (*) contain variable values. You can code any appropriate value in these fields. However, the other fields can contain only the values shown in the chart. Code these fields exactly as the chart specifies.

Field	Size Type	Explanation	What to Code
RMHBSCC	2 hex	Starts the message	RMHBSCC DATA X'1002'
RMHID	1 alpha	Identifies a message to $RMU	RMHID DATA C'X'
RMHTYP	1 alpha	Specifies a request to $RMU	RMHTYP DATA C'R'
RMREQ	2 num	Specifies RECEIVE request	RMREQ DATA F'1'
RMRDSN (*)	8 alpha	Name of data set to receive host data	RMRDSN DATA CL6'DATASET'
RMRVOL (*)	7 alpha	Volume containing data set to receive host data. Default=IPL volume	RMRVOL DATA CL6'VOLUME'
RMRSTR (*)	4 num	Starting record of host data set	RMRSTR DATA D'n'
RMRTYP (*)	2 num	Type of host data set to be received	RMRTYP DATA F'n'
RMRBLK (*)	2 num	Specifies record blocking	RMRBLK DATA F'n

Figure 38. Required Fields for RECEIVE Request

Here is an example of a RECEIVE request sent by a host program. The host sends a data set called "MYDATA" to volume "MYVOL" on the remote system.

Figure 54 on page CO-111 shows a sample program that sends a RECEIVE request.

Host Program	Host	Remote
BSCWRITE IX	ENQ ------->	
		<------- ACK*
RMHBSCC DATA X'1002'	TEXT ------->	
RMHID DATA C'X'		
RMHTYP DATA C'R'		
RMREQ DATA F'1'		
RMRDSN DATA CL8'MYDATA'		
RMRVOL DATA CL6'MYVOL'		
RMRSTR DATA D'0'		
RMRTYP DATA F'0'		
RMRBLK DATA F'1'		
		<------- ACK*
BSCWRITE E	EOT ------->	
		<------- ENQ
	ACK* ------->	
BSCREAD I		<------- TEXT
		(status)
		RMHTYP='S'
		RMSREQ=1
		RMSFN=-1
	ACK* ------->	
BSCREAD C		<------- EOT
	ENQ ------->	
BSCWRITE IX		<------- ACK*
RMHBSCC DATA X'1002'	TEXT ------->	
RMHID DATA C'X'		
RMHTYP DATA C'D'		
RMDDATA DATA C text		
		<------- ACK*
BSCWRITE C		
RMHBSCC DATA X'1002'	TEXT ------->	
RMHID DATA C'X'		
RMHTYP DATA C'D'		
RMDDATA DATA C text		
		<------- ACK*
BSCWRITE E	EOT ------->	
		<------- ENQ
	ACK* ------->	
BSCREAD I		<------- TEXT
		(count)
		RMHTYP='C'
		RMREQ=1
		RMCNT=2
	ACK* ------->	
BSCREAD C		<------- EOT

Figure 39. Communications Flow for RECEIVE

Remote System Sending Data to Host (SEND)

You can ask $RMU to send a remote system data set to the host with the SEND request.

Send this request by coding a BSCWRITE IX instruction, along with the required information fields. Upon receiving request, $RMU first checks that it can send the requested records. It then sends a status record to indicate its ability to perform the function. A status record of -1 (successful completion) indicates the SEND function will continue; otherwise it terminates.

If $RMU is sending a program data set to the host, it also sends the program's RLD count, the program size in bytes, the program entry point and the program load address.

After transmitting the last data record of the data set to the host, $RMU sends a count message to indicate the number of records it sent the host. The host should compare this number to the number of records it received to verify that it got all the records $RMU sent. The RMCFLG field of the count message is not used for the SEND function.

If an unrecoverable error occurs, such as a disk or diskette read error, $RMU sends the host a status message with the appropriate error code, and terminates the SEND function. The host can terminate the SEND function by coding a BSCWRITE E instruction, followed by a status message and another BSCWRITE E.

Specifying the Starting Record: You must tell $RMU to start sending data from a particular record in the data set. For example, you may want it to start at the first record or at any other record within the data set.

The RMSSTR field is where to code this information. If you enter the value '0' or '1' in this field, $RMU starts sending the first record in the data set. If you enter any other number, $RMU starts sending from that particular record.

Specifying Data Set Type: You must tell $RMU what type of data set to send. The RMSTYP field is where to code this information. Enter one of the following values in this field:

- '0' for a standard data set with 256-byte records
- '1' for a source data set with 80-byte records.

Specifying Record Blocking: You must tell $RMU whether or not to block the records it sends. If you want $RMU to block the records, you must specify the number of blocks in each record.

The RMSBLK field is where to code this information. Enter one of the following values in this field:

- '0' or '1' to specify no blocking
- any other number to specify the exact number of blocks for $RMU to send in one record.

Coding Required Fields for SEND Function

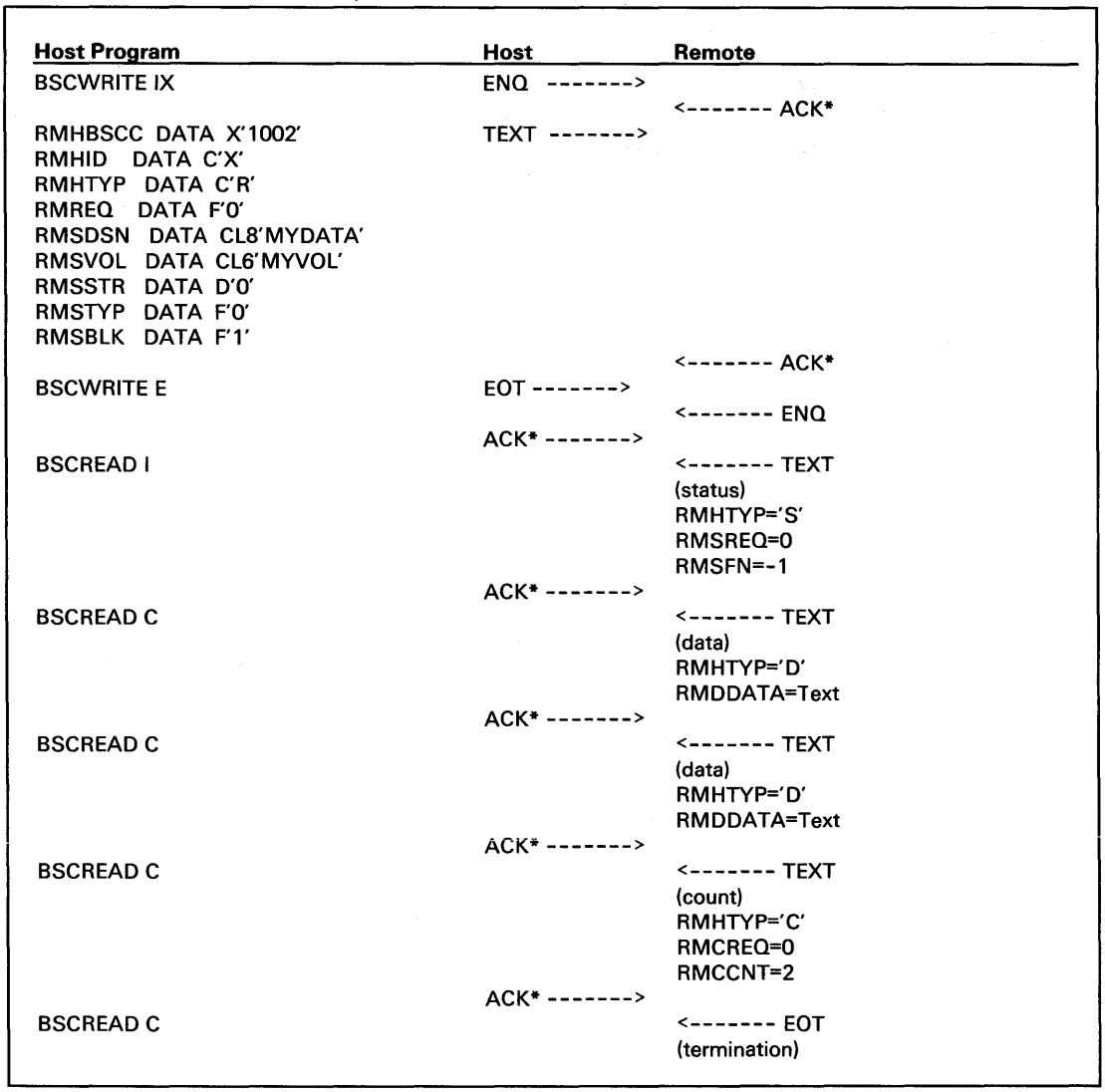
The chart that follows shows all the fields to code in your program for a SEND request. The fields identified with an asterisk (*) contain variable values. You can code any appropriate value in these fields. However, the other fields can contain only the values shown in the chart. Code these fields exactly as the chart specifies.

Field	Size Type	Explanation	What to Code
RMHBSCC	2 hex	Starts the message	RMHBSCC DATA X'1002
RMHID	1 alpha	Identifies a message to $RMU	RMHID DATA C'X'
RMHTYP	1 alpha	Identifies a request to $RMU	RMHTYP DATA C'R'
RMREQ	2 num	Specifies the SEND request	RMREQ DATA F'0'
RMSDSN (*)	8 alpha	Name of data set to send to host	RMSDSN DATA CL8'DATASET'
RMSVOL (*)	6 alpha	Volume containing the data set to be sent. Default=IPL volume	RMSVOL DATA CL6'VOLUME'
RMSSTR (*)	4 num	Starting record of the data set to be sent	RMSSTR DATA D'0'
RMSTYP (*)	2 num	Type of data set to be sent to the host	RMSTYP DATA F'0'
RMSBLK (*)	2 num	Specifies record blocking of data to be sent	RMSBLK DATA F'0'

Figure 40. Required Fields for SEND Request

Here is an example of SEND request communications flow. The host asks the remote system to send it a data set called "MYDATA."

Figure 55 on page CO-115 shows a sample program that sends a SEND request.



Host Program	Host	Remote
BSCWRITE IX	ENQ ------->	
		<------- ACK*
RMHBSCC DATA X'1002'	TEXT ------->	
RMHID DATA C'X'		
RMHTYP DATA C'R'		
RMREQ DATA F'0'		
RMSDSN DATA CL8'MYDATA'		
RMSVOL DATA CL6'MYVOL'		
RMSSTR DATA D'0'		
RMSTYP DATA F'0'		
RMSBLK DATA F'1'		
		<------- ACK*
BSCWRITE E	EOT ------->	
		<------- ENQ
	ACK* ------->	
BSCREAD I		<------- TEXT
		(status)
		RMHTYP='S'
		RMSREQ=0
		RMSFN=-1
	ACK* ------->	
BSCREAD C		<------- TEXT
		(data)
		RMHTYP='D'
		RMDDATA=Text
	ACK* ------->	
BSCREAD C		<------- TEXT
		(data)
		RMHTYP='D'
		RMDDATA=Text
	ACK* ------->	
BSCREAD C		<------- TEXT
		(count)
		RMHTYP='C'
		RMCREQ=0
		RMCCNT=2
	ACK* ------->	
BSCREAD C		<------- EOT
		(termination)

Figure 41. Communications Flow for SEND Request

Remote System Echoing Host Data (WRAP)

The host can send data to the remote system, and ask $RMU to echo that data back to the host. This is the WRAP request. WRAP is useful for testing line conditioning.

Send a WRAP request by coding a BSCWRITE IX instruction, along with the required information fields. The RMWTXT field is where to specify the text you want $RMU to echo back to the host.

Coding the Required Fields for WRAP Request

Specify the following fields for the WRAP function:

The chart that follows shows all the fields to code in your program for a WRAP request. The fields identified by as asterisk (*) contain variable values. You can code any appropriate value in these fields. However, the other fields can contain only the values shown in the chart. Code these fields exactly as the chart specifies.

Field	Size Type	Explanation	What to Code
RMHBSCC	2 hex	Starts the message	RMHBSCC DATA X'1002
RMHID	1 alpha	Identifies a message to $RMU	RMHID DATA C'X'
RMHTYP	1 alpha	Identifies a request to $RMU	RMHTYP DATA C'R'
RMREQ	2 num	Specifies the WRAP request	RMREQ DATA F'5'
RMWTXT (*)	vari- able	Text that $RMU is to echo back to host.	RMWTXT DATA C'ANY TEXT'

Figure 42. Required Fields for WRAP Request

Figure 43 on page CO-86 shows an example of the WRAP function. The host sends the Series/1 a WRAP request along with the text "WRAP TEXT." The Series/1 receives the request and transmits the identical data back to the host, and the operation is completed.

Figure 53 on page CO-109 shows a sample program that can send a WRAP request.

Host Program	Host	Remote
BSCWRITE IX	ENQ -------->	
		<-------- ACK*
RMHBSCC DATA X'1002'	TEXT ------->	
RMHID DATA C'X'		
RMHTYP DATA C'R'		
RMREQ DATA F'5'		
RMWTXT DATA C'WRAP TEXT'		
		<-------- ACK*
BSCWRITE E	EOT ------->	
		<-------- ENQ
	ACK* ------->	
BSCREAD I		<-------- TEXT
		(wrap text)
		RMHBSCC=X'1002'
		RMHID=C'X'
		RMHTYP=C'R'
		RMREQ=F'5'
		RMWTXT=X'WRAP TEXT'
	ACK* ------->	
BSCREAD C		<-------- EOT
		(termination)

Figure 43. Communications Flow for WRAP

Controlling Program Execution on the Remote System

$RMU can perform functions involving the execution of programs on the remote system. You can tell $RMU to start a program running on the remote system. You can also tell $RMU to load another program and at the same time terminate itself. The requests associated with these functions are:

- EXEC start a program on the remote system
- SHUTDOWN the operation of $RMU and load another program on the remote system.

Host Starting a Program on Remote System (EXEC)

The host can ask $RMU to start execution of a program on the remote system. This is the EXEC request.

Send the EXEC request by coding a BSCWRITE IX instruction, along with the required data fields.

$RMU sends a the host a status message to tell if it was able to perform the EXEC function. $RMU then waits for a new request from the host.

Coding the RMXFLG Field: The RMXFLG is an optional field which activates these conditions:

- Prints a "program loaded" message on the remote terminal that loaded $RMU. Enter the value X'40'.
- Causes $RMU to wait for the program to finish running before sending a status message to the host. Enter the value X'20'.

These two values correspond to the LOGMSG and WAIT operands of the Event Driven Language instruction LOAD.

To specify both of these conditions, enter the value X'60'.

To specify neither of these conditions, simply do not code the RMXFLG field.

Specifying Partition: You must tell $RMU the partition in which to run the program. The field RMXPTN is where to code this information. Enter one of the following values in this field:

- -1 $RMU partition
- 0 Any partition
- 1 - 8 Specific partition.

Allocating Free Space: You can specify the amount of free space (in bytes) to pass to the program. The RMXLFS field is where to code this information.

Passing Parameters: Some programs require parameters to be passed from the host in order to run successfully. You accomplish this with the RMXPRM# and RMXPRM fields.

In RMXPRM#, specify the length (in words) of the parameters to pass to the program.

In RMXPRM, specify the parameters themselves. The length of RMXPRM must be equal to the value in RMXPRM#.

Passing Data Sets: You can pass data sets to the program by coding the RMXDS# and RMXDS fields.

In RMXDS#, specify the number of data sets (up to nine) to pass to the program. Do not leave this field blank; enter zero if you are not passing any data sets.

In RMXDS, specify the name and volume of each data set to pass to the program. The number of RMXDS fields must be equal to the value of RMXDS#.

Coding Required Fields for EXEC Request

The chart that follows shows all the fields to code in your program for an EXEC request. The fields identified by as asterisk (*) contain variable values. You can code any appropriate value in these fields. However, the other fields can contain only the values shown in the chart. Code these fields exactly as the chart specifies.

Field	Size Type	Explanation	What to Code
RMHBSCC	2 hex	Starts the message	RMHBSCC DATA X'1002
RMHID	1 alpha	Identifies a message to $RMU	RMHID DATA C'X'
RMHTYP	1 alpha	Identifies a request to $RMU	RMHTYP DATA C'R'
RMREQ	2 num	Specifies the EXEC request	RMREQ DATA F'9'
filler	2	Reserved field (unused)	DATA F'0'
RMXFLG (*)	1 num	Load message prints (X'40') or status message is delayed (X'20') This field is optional.	RMXFLG DATA X'nn'
RMXPTN (*)	1 num	Partition to run program in	RMXPTN DATA H'n'
RMXPGM (*)	8 alpha	Name of program to execute	RMXPGM DATA CL8'DATASET'
RMXVOL (*)	6 alpha	Volume containing the program to execute. Default= IPL volume	RMXVOL DATA CL6'VOLUME'
RMXLFS (*)	2 num	Free space (in bytes) to pass to program	RMXLFS DATA F'nn'

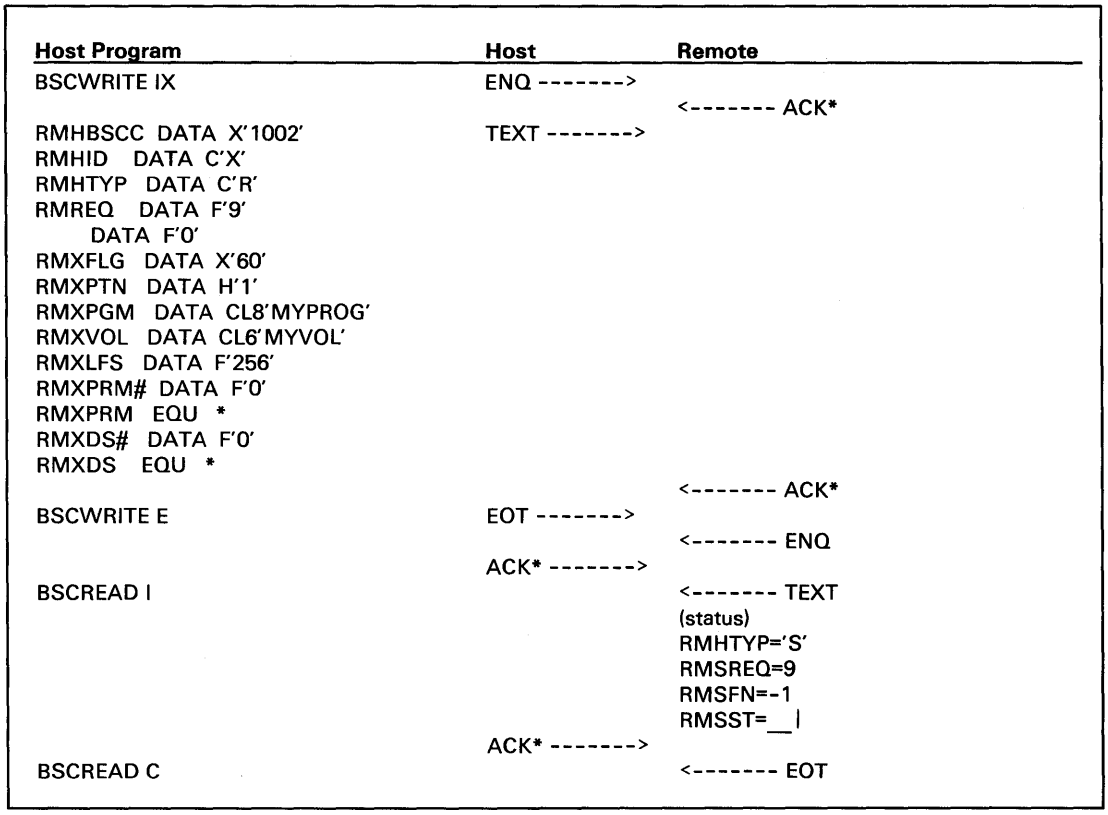
Figure 44 (Part 1 of 2). Required Fields for EXEC Request

Field	Size Type	Explanation	What to Code
RMXPRM# (*)	2 num	Length of parameters to pass to program	RMXPRM# DATA F'nn'
RMXPRM (*)	vari- able	Parameters to pass to program	RMXPRM EQU *
RMXDS# (*)	2 num	Number of data sets to pass to program	RMXDN# DATA F'nn'
RMXDS (*)	14 alpha	Name and volume of data set to pass to program.	RMXDS EQU *

Figure 44 (Part 2 of 2). Required Fields for EXEC Request

Figure 45 on page CO-90 shows an example of the EXEC function. The host specifies that a program named "MYPROG" on volume "MYVOL" is to be executed in partition 1, with 256 bytes of free space passed to the program. The RMXFLG field specifies that both the RMXFLGL and RMXFLGW bits are set on. No parameters or data sets are passed to "MYPROG." The program ends with a return code of -1. $RMU sends a status record of -1 (successful completion) to the host, along with the return code for "MYPROG."

Figure 53 on page CO-109 shows a sample program that can send an EXEC request.



Host Program	Host	Remote
BSCWRITE IX	ENQ ------->	
		<------- ACK*
RMHBSCC DATA X'1002'	TEXT ------->	
RMHID DATA C'X'		
RMHTYP DATA C'R'		
RMREQ DATA F'9'		
DATA F'0'		
RMXFLG DATA X'60'		
RMXPTN DATA H'1'		
RMXPGM DATA CL8'MYPROG'		
RMXVOL DATA CL6'MYVOL'		
RMXLFS DATA F'256'		
RMXPRM# DATA F'0'		
RMXPRM EQU *		
RMXDS# DATA F'0'		
RMXDS EQU *		
		<------- ACK*
BSCWRITE E	EOT ------->	
		<------- ENQ
	ACK* ------->	
BSCREAD I		<------- TEXT
		(status)
		RMHTYP='S'
		RMSREQ=9
		RMSFN=-1
		RMSST=__
	ACK* ------->	
BSCREAD C		<------- EOT

Figure 45. Communications Flow for EXEC

Terminate $RMU/Start Another Program on Remote System (SHUTDOWN)

You can ask $RMU to terminate and release any Series/1 resources it has allocated. In addition, you can ask $RMU to start another program on the remote system before terminating. This is the SHUTDOWN request.

Send your SHUTDOWN request by coding a BSCWRITE IX instruction, along with the required information fields. The request may also specify the name of a program to be executed, similar in format to the EXEC function.

Coding the RMSDFLG Field: The RMSDFLG is an optional field which activates these conditions:

- Specifies that $RMU is to run another program. Enter the value X'80'.
- Prints a "program loaded" message on the remote terminal that loaded $RMU. Enter the value X'60'. This value corresponds to the LOGMSG parameter of the Event Driven Language instruction LOAD.

To specify both conditions, enter the value X'CO'

To specify neither of these conditions, simply do not code the RMSDFLG field.

Specifying Partition: You must tell $RMU the partition in which to run the program. The field RMSDPTN is where to code this information. Enter one of the following values in this field:

- -1 $RMU partition
- 0 Any partition
- 1 - 8 Specific partition.

Allocating Free Space: You can specify the amount of free space (in bytes) to pass to the program. The RMSDLFS field is where to code this information. This value is expressed in bytes.

Passing Parameters: Some programs require parameters to be passed from the host system in order to run successfully. You accomplish this with the RMSDPRM# and RMSDPRM fields.

In RMSDPRM#, specify the length (in words) of the parameters to pass to the program.

In RMSDPRM, specify the parameters themselves. The length of RMSDPRM must be equal to the value in RMSDPRM#.

Passing Data Sets: You can pass data sets to the program by coding the RMSDDS# and RMSDDS fields.

In RMSDDS#, specify the number of data sets (up to nine) to pass to the program. Do not leave this field blank; enter zero if you are not passing any data sets.

In RMSDDS, specify the name and volume of each data set to pass to the program. The number of RMSDDS fields must be equal to the value of RMSDDS#.

Coding the Required Fields for SHUTDOWN Request

The chart that follows shows all the fields to code in your program for a SHUTDOWN request. The fields identified by an asterisk (*) contain variable values. You can code any appropriate value in these fields. However, the other fields can contain only the values shown in the chart. Code these fields exactly as the chart specifies.

Field	Size Type	Explanation	What to Code
RMHBSCC	2 hex	Starts the message	RMHBSCC DATA X'1002'
RMHID	1 alpha	Identifies a message to $RMU	RMHID DATA C'X'
RMHTYP	1 alpha	Identifies a request to $RMU	RMHTYP DATA C'R'
RMREQ	2 num	Specifies the SHUTDOWN request	RMREQ DATA F'7'
filler	2	Reserved field (unused)	DATA F'0'
RMSDFLG (*)	1 num	Execute another program (X'80') and/or load message prints (X'40') This field is optional.	RMSDFLG DATA X'nn'
RMSDPTN (*)	1 num	Partition to run program in	RMSDPTN DATA H'n'
RMSDPGM (*)	8 alpha	Name of program to execute	RMSDPGM DATA CL8'DATASET'
RMSDVOL (*)	6 alpha	Volume containing the program to execute. Default= IPL volume	RMSDXVOL DATA CL6'VOLUME'
RMSDLFS (*)	2 num	Free space (in bytes) to pass to program	RMSDLFS DATA F'nn'

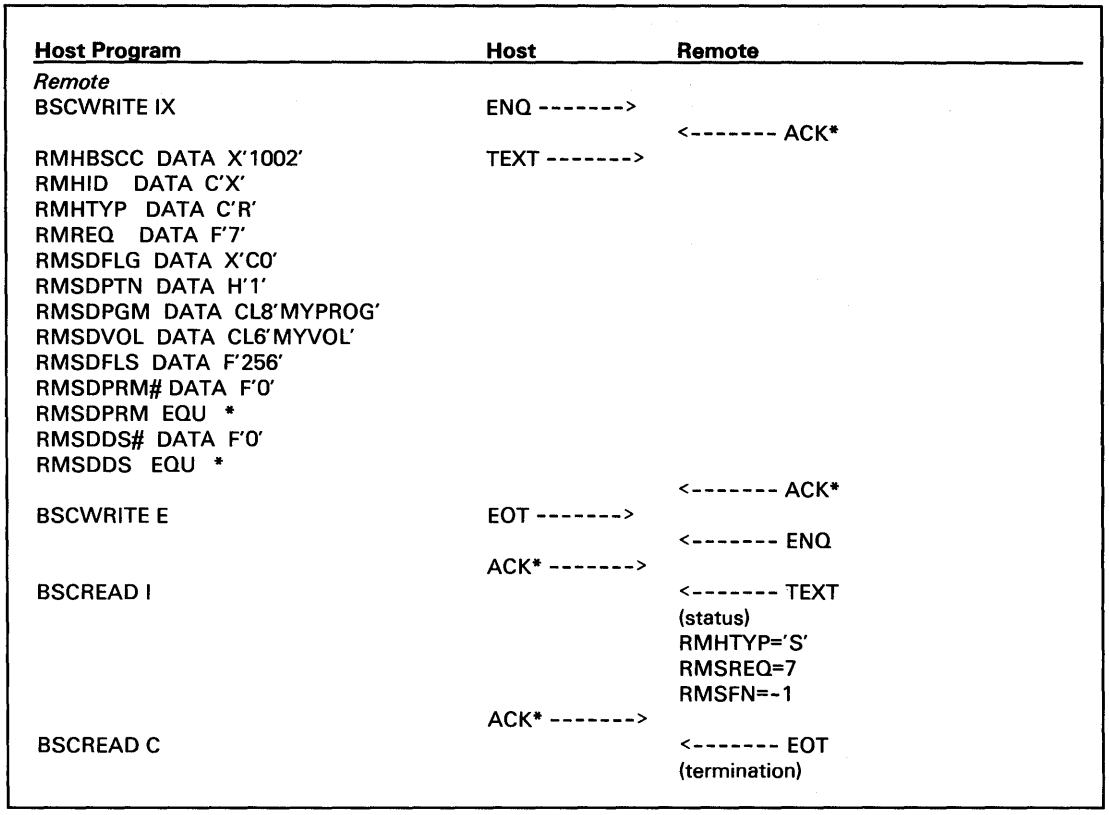
Figure 46 (Part 1 of 2). Required Fields for SHUTDOWN Request

Field	Size Type	Explanation	What to Code
RMSDPRM# (*)	2 num	Length of parameters to pass to program	RMSDPRM# DATA F'nn'
RMSDPRM (*)	vari- able	Parameters to pass to program	RMSDPRM EQU *
RMSDDS# (*)	2 num	Number of data sets to pass to program	RMSDDN# DATA F'nn'
RMSDDS (*)	14 alpha	Name and volume of data set to pass to program.	RMSDDS EQU *

Figure 46 (Part 2 of 2). Required Fields for SHUTDOWN Request

Figure 47 on page CO-94 shows an example of the SHUTDOWN function. The host sends the Series/1 a SHUTDOWN request with a program name specified. The program, "MYPROG" on volume "MYVOL," is to execute in partition 1, has 256 bytes of free space passed to it, and has no parameters or data sets passed to it. The RMSDFLG field specifies that a program is to be executed and a "program loaded" message is to be printed following a successful load of the program. $RMU sends a status record of -1 (successful completion) to the host, loads the program, and $RMU ends.

Figure 53 on page CO-109 shows a sample program that can send a SHUTDOWN request.



Host Program	Host	Remote
Remote		
BSCWRITE IX	ENQ ------->	
		<------- ACK*
RMHBSCC DATA X'1002'	TEXT ------->	
RMHID DATA C'X'		
RMHTYP DATA C'R'		
RMREQ DATA F'7'		
RMSDFLG DATA X'C0'		
RMSDPTN DATA H'1'		
RMSDPGM DATA CL8'MYPROG'		
RMSDVOL DATA CL6'MYVOL'		
RMSDFLS DATA F'256'		
RMSDPRM# DATA F'0'		
RMSDPRM EQU *		
RMSDDS# DATA F'0'		
RMSDDS EQU *		
		<------- ACK*
BSCWRITE E	EOT ------->	
		<------- ENQ
	ACK* ------->	
BSCREAD I		<------- TEXT
		(status)
		RMHTYP='S'
		RMSREQ=7
		RMSFN=-1
	ACK* ------->	
BSCREAD C		<------- EOT
		(termination)

Figure 47. Communications Flow for SHUTDOWN

Verifying Identities between Systems (IDCHECK)

You can ask $RMU to verify the identities of the host and remote systems. This is the IDCHECK request. You send your IDCHECK request by coding a BSCWRITE IX instruction along with the required information fields.

The IDs of both the host and remote systems have predefined default values. These values are discussed in the section, "Remote Management Utility Defaults." The default IDs are in remote system storage. $RMU asks the host to send its ID for verification. Only if the host ID is correct does $RMU send the remote system ID to the host. If the host sends an incorrect (invalid) ID, $RMU terminates the function.

Coding the Required Fields for IDCHECK Function

The chart that follows shows all the fields to code in your program for an IDCHECK request. The fields identified by an asterisk (*) contain variable values. You can code any appropriate value in these fields. However, the other fields can contain only the values shown in the chart. Code these fields exactly as the chart specifies.

Field	Size Type	Explanation	What to Code
RMHBSCC	2 hex	Starts the message	RMHBSCC DATA X'1002
RMHID	1 alpha	Identifies a message to $RMU	RMHID DATA C'X'
RMHTYP	1 alpha	Identifies a request to $RMU	RMHTYP DATA C'R'
RMREQ	2 num	Specifies the IDCHECK request	RMREQ DATA F'9'
RMICHK (*)	8 alpha	Host ID	RMICHK DATA C'HOSTID'

Figure 48. Required Fields for IDCHECK Request

Figure 49 shows an example of the IDCHECK function. The host sends the default ID "HOSTRMUX." $RMU validates the host ID and sends a status record of -1 (successful completion) to the host along with its default ID, "REMTRMUX."

Figure 53 on page CO-109 shows a sample program that can send an IDCHECK request.

```
Host Program                Host              Remote
BSCWRITE IX                 ENQ ------->
                                              <------- ACK*
RMHBSCC DATA X'1002'        TEXT ------->
RMHID   DATA C'X'
RMHTYP  DATA C'R'
RMREQ   DATA F'6'
RMICHK  DATA C'HOSTRMUX'
                                              <------- ACK*
BSCWRITE E                  EOT ------->
                                              <------- ENQ
                            ACK* ------->
BSCREAD I                                     <------- TEXT
                                              (status)
                                              RMHTYP='S'
                                              RMSREQ=6
                                              RMSFN=-1
                                              RMSRID='REMTRMUX'
                            ACK* ------->
BSCREAD C                                     <------- EOT
```

Figure 49. Communications Flow for IDCHECK

Remote Management Utility ($RMU)

Interacting Between Host and Remote Systems (PASSTHRU)

You can set up interactive sessions between the host and remote systems with the PASSTHRU request. During a passthru session, the host can perform the same functions as a station directly attached to the remote system. The host can interact with the Event Driven Executive supervisor by issuing operator commands, or with a program or utility on the remote system.

Most programs that do not require full screen terminal support, including most Event Driven Executive utilities, are available for use during a passthru session. Programs which cannot be run under the PASSTHRU function are discussed in "Considerations for Using PASSTHRU."

Considerations for Using PASSTHRU

Certain restrictions apply to programming for the PASSTHRU session. Before establishing a session, you must know what you can and cannot do with PASSTHRU.

Virtual Terminal Support

PASSTHRU uses the virtual terminal support of the Event Driven Executive. Because of this, the restrictions inherent in virtual terminal support also apply to PASSTHRU. Virtual terminals do not support static screens. Therefore, programs that use static screens cannot be run under the PASSTHRU function. This includes programs such as the full screen editor, $FSEDIT. Another virtual terminal restriction is that the maximum record length is 254 bytes.

During system generation, you must define two virtual terminals for the remote system: CDRVTA and CDRVTB. You may want to change the LINSIZE parameter. A LINSIZE of 132 will handle output that uses the full width of a printer. Specifying a smaller value saves storage, but messages longer than the LINSIZE will be truncated. The maximum value for LINSIZE is 254.

Because the $RMU PASSTHRU function uses a predefined set of virtual terminals (CDRVTA and CDRVTB), only one PASSTHRU session can be conducted at a time. While a PASSTHRU session is being conducted, another copy of $RMU (defined for another communications line) can be performing any other function except PASSTHRU.

No Attention Interrupt

$RMU allows the host to transmit a Program Function key or an attention key only after $RMU has sent a request message. Therefore, when the attention key is pressed at a host terminal, the terminal stops communicating with $RMU (the remote system) and begins communicating with its own (the host) system. This prevents output from $RMU to the host from being interrupted by a host terminal attention key, as it could be by a local terminal. For example, a listing produced by the $DISKUT2 utility could not be interrupted by pressing the host terminal attention key and entering the $C command.

Deadlock and $RMUPA

A program that stops communicating with the terminal which loaded it and waits for operator commands (using the attention or Program Function key) will not run directly under the PASSTHRU function. This is because $RMU waits indefinitely on a READTEXT to the virtual channel at the same time the host program is waiting for an Attention or PF key. Since both programs are "listening" and neither is "talking," both will wait forever. This is called a deadlock. Programs that may do this include:

$DEBUG
$TRAP
$LOG
$BSCTRCE
$TERMUT3 (attention-entered commands)
$IOTEST (attention-entered commands)
CALCDEMO (sample program)

$RMUPA is a program that can break this deadlock. It must be started under the PASSTHRU function prior to starting a PASSTHRU session with a program which may have this problem. $RMUPA causes a "disconnect", which results in a $RMU sending a Program End PASSTHRU record to the host whenever the following events occur:

- No activity has occurred over the virtual channel for 20 seconds.
- $RMU is waiting on completion of a READTEXT instruction.
- The host program is not enqueued (ENQT) on its virtual terminal.

$RMUPA uses the STIMER instruction; therefore, timer support must be included in the Event Driven Executive system.

The sample PASSTHRU host program in "PASSTHRU Sample Program" on page CO-117 shows how to use $RMUPA. First $RMUPA is started. When a Program End PASSTHRU record is received at the host, the host responds with a Program End PASSTHRU record and the PASSTHRU session with $RMUPA ends. Only one copy of $RMUPA should be running at any time; it can run in any partition. It continues running until an "attention" followed by $RMUPA is entered.

Once $RMUPA is running, you may start another program. The sample PASSTHRU host program in "Example of Conducting a PASSTHRU Session" on page CO-125 shows how you can use $DEBUG. Note that you can enter $PF0 to provide the same function as the attention key.

If a remote program does not perform any terminal I/O for 20 seconds, $RMUPA causes a Program End record to be sent even though the program is still running. If this happens, the host should respond with a Request for Data record until the remote program performs terminal I/O.

Indefinite Waits

If the PASSTHRU function loads a program which issues an ENQT for a terminal other than the terminal which loaded $RMU and the program terminates, $RMU does not receive a "disconnect" over the virtual channel and the host will not receive a Program End record. $RMU will wait indefinitely. One example of when this occurs is when $EDXASM is running with output directed to a printer. This condition can be avoided in two ways:

- Load the program from another program (such as the $JOBUTIL utility) which will wait for the program to complete. Programs that require interaction with the terminal operator, such as $EDXASM, should be handled in this way.

- Load the program through a session with the Event Driven Executive supervisor (using the $L command) and respond with a Program End when the command terminates.

Abrupt Termination

If a PASSTHRU session is abruptly terminated (status received from host, invalid message received from host, or an error in the BSCAM), $RMU sends a return code 5 ("disconnected") to the program for the outstanding terminal request. This code will be received only once by the PASSTHRU-invoked program. The program should take appropriate action, which would most likely be to terminate.

If the program does not recognize the error and continues to perform terminal I/O, it will interfere with attempts to establish a new PASSTHRU session. If the new session is being established with a program, $RMU returns the status "virtual terminal busy." The host may establish a session with the Event Driven Executive supervisor and issue a $C command to cancel the suspended program. (As noted in the *Operator Commands and Utilities Reference*, the $C command should be used with caution).

When a load command ($L) is issued during a PASSTHRU session with the Event Driven Executive supervisor, a Program End record, resulting from completion of the command, may be received by the host. Whether it is received depends on how quickly the loaded program begins performing terminal I/O.

Timeouts

$RMU will not time-out while it is receiving messages during a PASSTHRU session. However, if the host does not acknowledge receipt of messages sent by $RMU, a time-out will occur and the PASSTHRU session will terminate. This can be avoided in two ways:

- Avoid any long delays at the host while messages are being received from the Series/1.

- Define a high retry count for the RETRIES parameter of the BSCLINE statement.

Send your PASSTHRU request by coding a BSCWRITE IX instruction, along with the required information fields. Once the passthru session begins, the host and $RMU exchange a series of messages in a manner similar to the way messages are written to and read from a terminal. This record exchange consists of two parts:

- Establishing a PASSTHRU session
- Conducting a PASSTHRU session .

Establishing a PASSTHRU Session

The host initiates the PASSTHRU function by sending a PASSTHRU request to $RMU. After the host receives a successful status record and an EOT, a *PASSTHRU session* is established. The PASSTHRU request specifies (in the RMPRPGM field) the type of session:

- Communication with the Event Driven Executive supervisor
- Communication with a program or utility which $RMU will load .

If a session with the EDX supervisor is established, $RMU issues an "attention" (as if the attention key on the terminal were pressed). After the terminal on the host receives the caret symbol (>), the host operator can enter a Series/1 operator command, for example, $L.

If a session with a program is established, the host specifies the name of the program and $RMU loads the program. The PASSTHRU session will be conducted with the host interacting with the program.

Coding Required Fields for PASSTHRU Request

The chart that follows shows all the fields to code in your program for a PASSTHRU request. The fields identified by an asterisk (*) contain variable values. You can code any appropriate value in these fields. However, the other fields can contain only the values shown in the chart. Code these fields exactly as the chart specifies.

Field	Size Type	Explanation	What to Code
RMHBSCC	2 hex	Starts a message	RMHBSCC DATA X'1002'
RMHID	1 alpha	Identifies a message to $RMU	RMHID DATA C'X'
RMHTYP	1 alpha	Identifies a request	RMHTYP DATA C'R'

Figure 50 (Part 1 of 2). Required Fields for PASSTHRU Request

Field	Size Type	Explanation	What to Code
RMREQ	2 num	Specifies the PASSTHRU request	RMREQ DATA F'12'
RMPRBLK*	2 num	Specifies record blocking by remote system	RMPRBLK DATA F'n'
RMPRFLG	1	Reserved field (unused)	RMPFLG DATA H'0'
RMPRPTN*	1 num	Partition to run program or utility in	RMPRPTN DATA H'n'
RMPRPGM*	8 alpha	Name of program or utility to interact with host	RMPRPGM DATA CL8'PROGNAME'
RMPRVOL*	6 alpha	Volume containing the program or utility Default= IPL volume	RMPRVOL DATA CL6'VOLUME'
RMPRLFS*	2 num	Free space to pass to program	RMPRLFS DATA F'n'
RMPRPRM# (*)	2 num	Length of parameters to pass to program	RMPRPRM# DATA F'n'
RMPRPRM*	vari-able	Parameters to be passed to the program.	RMPRPRM EQU *
RMPRDS# (*)	2 num	Number of data sets to pass to program	RMPRDS# DATA F'n'
RMPRDS	14 alpha	Name and volume of data sets to pass to program	RMPRDS EQU *

Figure 50 (Part 2 of 2). Required Fields for PASSTHRU Request

Here is an example of the communications flow for a PASSTHRU request.

Figure 56 on page CO-118 shows a sample program that sends a PASSTHRU request.

Host Program	Host	Remote
BSCWRITE IX	ENQ ------->	
		<------- ACK*
RMHBSCC DATA X'1002'	TEXT ------->	
RMHID DATA C'X'		
RMHTYP DATA C'R'		
RMREQ DATA F'12'		
RMPRFLG DATA H'0'		
RMPRPTN DATA H'0'		
RMPRPGM DATA CL8'MYPROG'		
RMPRVOL DATA CL6'MYVOL'		
RMPRLFS DATA F'256'		
RMPRBLK DATA F'0'		
RMPRPRM# DATA F'0'		
RMPRPRM EQU *		
RMPRDS# DATA F'0'		
RMPRDS EQU *		
		<------- ACK*
BSCWRITE E	EOT ------->	
		<------- ENQ
	ACK* ------->	
BSCREAD I		<------- TEXT
		(status)
		RMHTYP ='S'
		RMSREQ=12
		RMSFN=-1
	ACK* ------->	
BSCREAD C		<------- EOT
		<------- ENQ
	ACK* ------->	
BSCREAD I		<------- TEXT
		(passthru data)
		RMHTYP='P'
		RMPTYP=1
		RMPST=Status from READTEXT
		RMPTXTL=Message length
		RMPTXT=Message text
	ACK* ------->	

Figure 51 (Part 1 of 2). Communications Flow for PASSTHRU

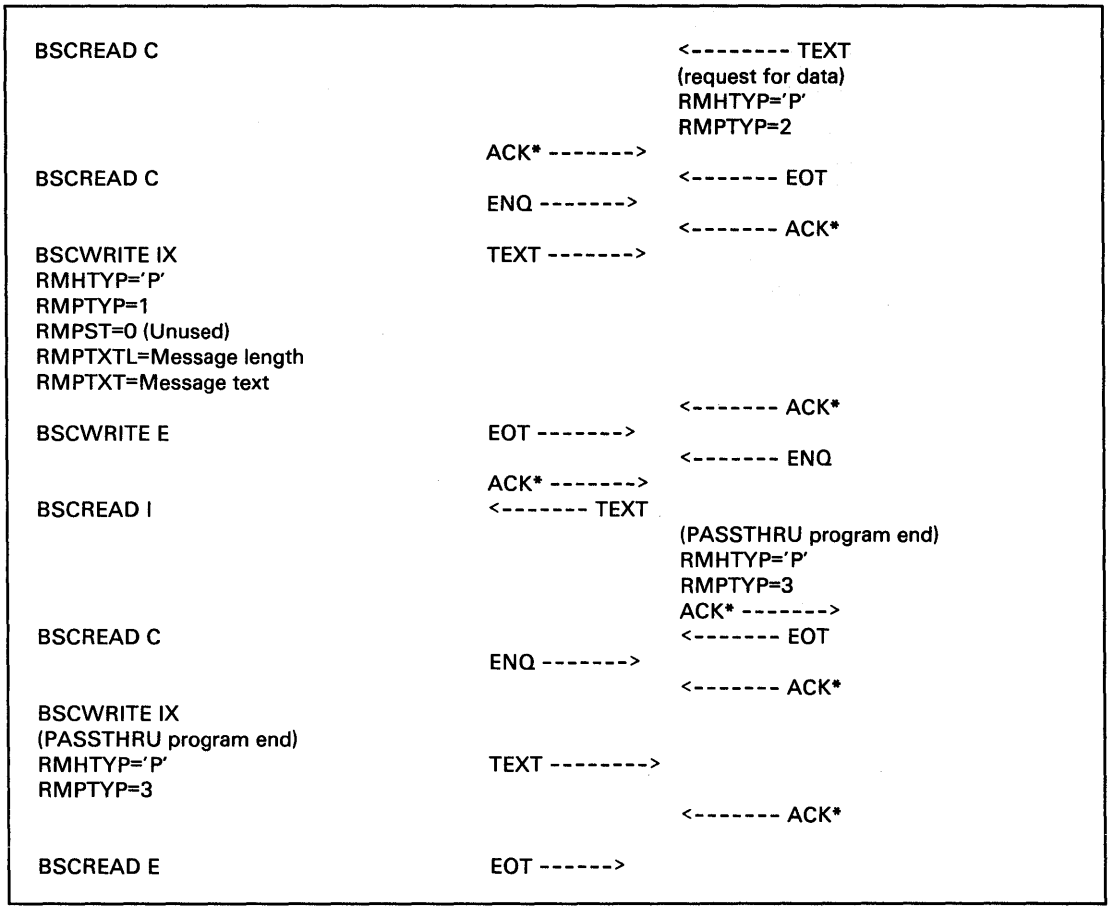


Figure 51 (Part 2 of 2). Communications Flow for PASSTHRU

Conducting a PASSTHRU Session

Once the PASSTHRU session is established, the host and the remote systems exchange PASSTHRU records. These records provide information to and receive information from the host program, as if the host program were a terminal on the remote system. There are four types of PASSTHRU records:

- Text or Program Function (PF) Key - passes messages or Program Function keys
- Request for Data - indicates data should be sent
- Program End - indicates termination
- No Data - indicates no messages are available.

The host "state" can be changed by:

- Receiving a PASSTHRU record from $RMU. This is shown as a solid horizontal line with an arrow pointing to the new state.
- Sending a PASSTHRU record to $RMU. This is shown as a horizontal line of dashes with an arrow pointing to the new state.
- A change of state with no PASSTHRU record transfer. This is represented by a dotted line with an arrow pointing to the new state.

The PASSTHRU session begins with the host in the state READTEXT. The host issues a read to the communications line and receives either a Text or PF Key, Request for Data, or Program End record.

If the host receives a Text or PF Key record, the Series/1 is sending data to the host. The program (or the supervisor) has issued a PRINTEXT or other terminal I/O instruction, and it is transmitted to the host as if the host were a terminal. The state of the host changes from READTEXT to READING, the host reads the Text or PF Key record, and the state then changes back to READTEXT. The host remains in the READTEXT state as long as it receives Text or PF Key records.

If the host receives a Request for Data record, $RMU needs data from the host. The program (or the supervisor) has issued a READTEXT or other terminal I/O instruction, and requires data from the host as if the host were a terminal. The state of the host changes from READTEXT to "PGM NEEDS DATA". Note that an EOT follows the the Request for Data record. The host must also read the EOT.

When the host is in the state "PGM NEEDS DATA," it must send a Text or PF Key record followed by an EOT. The Text or PF Key record the host sends can contain either text or a PF key.

If the host sends text, the state of the host changes from "PGM NEEDS DATA" to READTEXT. If the host sends a Program Function key, the host goes to the state "PFK SENT." The host issues a read to the communications line and will receive a Request for Data record followed by an EOT. $RMU sends the Request for Data record to the host because the original request was not satisfied by the Program Function key. As a result, the host is now in the state "SEND TEXT." The host must send a Text or PF Key record which contains text, followed by an EOT. The host then returns to the state READTEXT.

If the host is in the state READTEXT and receives a Program End record followed by an EOT, this means that the program, the operator command, or an attention exit has completed. The host changes from the state READTEXT to "CONTINUE ?". At this point, the host must determine whether the PASSTHRU session should continue.

If the PASSTHRU session is with a program and the program ends (while in the "CONTINUE ?" state), the host usually does not continue the session. If the session is with the supervisor and you enter a $L command, the host usually continues the session and communicates with the program that was loaded.

To terminate the PASSTHRU session, the host sends a Program End record, followed by an EOT. This changes the state of the host from "CONTINUE ?" to "EXIT". The PASSTHRU session now terminates and the Remote Management Utility waits for a new request from the host. To continue the session, the host sends a Request for Data record followed by an EOT. The state of the host then changes from "CONTINUE ?" to "ACTIVITY ?".

At this point, $RMU determines if there is any activity on the Series/1 for the host. If there is, $RMU sends one of the three PASSTHRU records (Text or PF Key, Request for Data, or Program End) which the host can receive in the "READTEXT" state. The state of the host then changes depending on the type of PASSTHRU record it receives.

If there is no terminal activity, $RMU sends a No Data record followed by an EOT, and the host state changes from "ACTIVITY ?" to "CONTINUE ?". The host then determines whether it should continue. If the program in the Series/1 has delays in performing terminal I/O while the host is in the "CONTINUE ?" state, the host may change from "CONTINUE ?" to "ACTIVITY ?" and back again several times. However, if no activity occurs, the host must eventually send a Program End record and terminate the PASSTHRU session.

Figure 57 on page CO-125 shows a PASSTHRU session that invokes and runs the $DEBUG utility from the host terminal.

PASSTHRU Record Types

This section describes the format and content of the four types of PASSTHRU records.

Text or Program Function Key Record

This record consists of two segments. The first six bytes, or the main segment, identifies the record as a PASSTHRU Text or Program Function (PF) key record. One or more text or PF key segments follow the main segment.

In the main segment, all values are constants, as shown below. The number 1 for the RMPTYP field identifies the record as a text or PF key record. The text or Program Function key segment contains the information to be transferred.

```
Main segment:

 RMHBSCC DATA X'1002'
 RMHID   DATA C'X'
 RMHTYP  DATA C'P'
 RMPTYP  DATA F'1'

Text or Program Function key segment:

 RMPST   DATA F'nnnn'
 RMPTXTL DATA F'nnnn'
 RMPTXT  DATA C'xxxx'
```

The fields in the text or Program Function key segment are:

RMPST A 2-byte numeric field containing the return code associated with the text. For example, the return code might indicate that the text is to appear on a new line. This field contains a value only on records received by the host.

Some return codes have no text associated with them. For a complete description of the possible return codes, see the virtual terminal return codes for the READTEXT instruction in the *Language Reference*.

The return codes which apply are:

X'8Fnn'	LINE=nn received
X'8Enn'	SKIP=nn received
-2	Line received (no CR)
-1	New line received

RMPTXTL A 2-byte numeric field specifying either the length of the text, or indicating a PF key is being sent (-1). If there is no text, for example when only a return code is sent, this field contains a zero.

RMPTXT Either a variable-length alphameric field containing text, or a 2-byte numeric field containing the PF key value. If the field contains text, the length of the text must equal RMPTXTL. If RMPTXTL is an odd number, one byte of blanks (X'40') follows the text.

If the Text or PF Key record is not blocked, it will contain one of each segment. If the record is blocked, it will contain one main segment followed by multiple text or PF key segments. The host must determine the length of the record to process each segment. All records sent by the host are unblocked. Records sent by $RMU may be blocked if specified on the PASSTHRU request. Blocking is discussed in "PASSTHRU Blocking" on page CO-108.

All Text or PF Key records sent by $RMU will always contain text; the host will never receive a Program Function key.

When the host sends a Text or PF Key record, the record may contain either text (the host as a terminal has entered text), or a PF key (the host as a terminal has entered a PF key). If text is sent, the length of the text is specified in the RMPTXTL field, and the text is specified in the RMPTXT field. The RMPST field is not used.

The following example shows a record sent by the host which contains the text "MESSAGE FROM HOST PROGRAM."

```
RMHBSCC DATA X'1002'
RMHID   DATA C'X'
RMHTYP  DATA C'P'
RMPTYP  DATA F'1'
RMPST   DATA F'0'  (IGNORED)
RMPTXTL DATA F'25'
RMPTXT  DATA C'MESSAGE FROM HOST PROGRAM'
```

When the host sends a PF key, the value of the RMPTXTL field is set to -1 and the PF key is specified as a 2-byte numeric value in the RMPTXT field. A PF key value of 0 is the equivalent of an "attention."

The following example shows the host sending a Program Function key 3.

```
RMHBSCC DATA X'1002'
RMHID   DATA C'X'
RMHTYP  DATA C'P'
RMPTYP  DATA F'1'
RMPST   DATA F'0'  (IGNORED)
RMPTXTL DATA F'-1' (INDICATES PF KEY)
RMPPF   DATA F'3'  PF KEY 3
```

Figure 52 on page CO-107 is an example of the records the host receives from a program which executes a PRINTEXT instruction.

```
Issued by program on Series/1:

 PRINTEXT 'ENTER COMMAND',SKIP=1


PASSTHRU record received by host
with no blocking:

 RMHBSCC DATA X'1002'
 RMHID   DATA C'X'
 RMHTYP  DATA C'P'
 RMPTYP  DATA F'1'
 RMPST   DATA X'8E01'  (SKIP=1)
 RMPTXTL DATA F'0'     (NO TEXT)

 RMHBSCC DATA X'1002'
 RMHID   DATA C'X'
 RMHTYP  DATA C'P'
 RMPTYP  DATA F'1'
 RMPST   DATA F'-2'
 RMPTXTL DATA F'13'
 RMPTXT  DATA C'ENTER COMMAND'
      DATA C' '     (PAD)

PASSTHRU record received by host
with blocking:

 RMHBSCC DATA X'1002'
 RMHID   DATA C'X'
 RMHTYP  DATA C'P'
 RMPTYP  DATA F'1'
      DATA X'8E01'  (SKIP=1)
      DATA F'0'     (NO TEXT)
      DATA F'-2'    (NEXT SEGMENT)
      DATA F'13'
      DATA C'ENTER COMMAND'
      DATA C' '     (PAD)
```

Figure 52. Example of PASSTHRU Records Received by Host

Request for Data Record

The Request for Data record is a 6-byte record that contains constant values. A Request for Data record is always followed by an EOT. The format of the Request for Data record is:

```
RMHBSCC DATA X'1002'
RMHID   DATA C'X'
RMHTYP  DATA C'P'
RMPTYP  DATA F'2'
```

Program End Record

The Program End record is a 6-byte record that contains constant values. A Program End record is always followed by an EOT. The format of the Program End record is:

```
RMHBSCC DATA X'1002'
RMHID   DATA C'X'
RMHTYP  DATA C'P'
RMPTYP  DATA F'3'
```

No Data Record

The No Data record is a 6-byte record that contains constant values. A No Data record is always followed by an EOT. The format of the No Data record is:

```
RMHBSCC DATA X'1002'
RMHID   DATA C'X'
RMHTYP  DATA C'P'
RMPTYP  DATA F'4'
```

PASSTHRU Blocking

When PASSTHRU records are not blocked, each Text or PF Key record contains only one text segment. With blocking, each record may contain multiple text segments. For PASSTHRU sessions in which the host receives many consecutive lines of output, such as a result of a "list" command to a utility, blocking allows more efficient usage of the communications line.

The host specifies blocking in the RMPRBLK field of the PASSTHRU request. If this field is zero, blocking is not performed. A value greater than zero indicates the maximum size, in bytes, of the text segments which the host can process.

To determine the value for the RMPRBLK field, start with the size of the buffer at the host. Subtract 6 from the size of the host buffer for the 6-byte main segment of each record. Then subtract 2 more to allow space for the ETX plus one byte for word alignment. The resulting number is the maximum block size.

$RMU will use this value if it can. However, if $RMU does not have a buffer as large as the value of RMPRBLK, $RMU will use the largest block size it can.

The host must determine the length of the Text or PF Key record and process each text segment until the end of the record is reached. If a text record exceeds the block size specified in RMPRBLK, $RMU still sends that record to the host. This may result in a "wrong length record" condition. The host should ensure that it can handle the longest length record expected from the utility. For example, if the longest text record is 132 bytes, a block size of 136 would be sufficient for all records.

Sample Programs

The following sample Series/1 programs communicate with and perform functions of the Remote Management Utility.

Multifunction Program

This program executes on a host Series/1 and communicates with $RMU on a remote Series/1. The program performs all the functions of $RMU except SEND, RECEIVE, and PASSTHRU. The program sends an ALLOCATE request and prints a status message, but can be used for the other functions by simply defining the fields of the desired request at label "RM."

```
UT        PROGRAM START
START     EQU       *
          BSCOPEN   IOCB,ERROR=BSCERR      OPEN BSC LINE
          MOVE      IOCB3,+REQLEN          LENGTH OF REQUEST
*                                          IN IOCB
          BSCWRITE  IX,IOCB,ERROR=BSCERR   WRITE REQUEST
          BSCWRITE  E,IOCB,ERROR=BSCERR    WRITE EOT
          MOVEA     IOCB2,ST               ADDRESS OF STATUS
          MOVE      IOCB3,20               LENGTH OF STATUS
*                                          IN IOCB
          BSCREAD   I,IOCB,ERROR=BSCERR,TIMEOUT=NO  READ STATUS
          SUB       IOCB,IOCB2,RESULT=PN2 LENGTH INTO PRINTNUM
          ADD       PN2,+1
          SHIFTR    PN2,1                  CONVERT LENGTH TO WORDS
          PRINTEXT  '@STATUS MESSAGE:@'
          PRINTNUM  ST,0,MODE=HEX,P2=PN2   PRINT STATUS MSG
          BSCREAD   C,IOCB,ERROR=BSCERR,TIMEOUT=NO   READ EOT
          IF        (ST+6,EQ,-1)           IF SUCCESSFUL STATUS
*                                          THEN
            PRINTEXT '@FUNCTION SUCCESSFUL'
          ELSE                              ELSE
            PRINTEXT '@FUNCTION FAILED'
          ENDIF                             ENDIF
TERM      EQU       *                      TERMINATION POINT
          BSCCLOSE  IOCB                   CLOSE BSC LINE
          PROGSTOP
*
BSCERR    EQU       *                    BSC ERROR ROUTINE
          MOVE      ST,UT                  MOVE RETURN CODE
          PRINTEXT  '@BSC ERROR:'
          PRINTNUM  ST                     PRINT RETURN CODE
          GOTO      TERM                   GO TO TERMINATION
```

Figure 53 (Part 1 of 2). Multifunction Sample Program

```
IOCB     BSCIOCB  9,RM,0,P2=IOCB2,P3=IOCB3     IOCB
*                              P2=IOCB2 IS MESSAGE ADDRESS
*                              P3=IOCB3 IS MESSAGE LENGTH
*
ST       DATA     10F'0'       AREA FOR STATUS RECORD
*                                10 BYTES  NORMAL STATUS RECORD
*                                 8 BYTES  IDCHECK STATUS EXT.
*                                 1 BYTE   ETX
*                                --------
*                                19 BYTES  TOTAL, ROUNDED UP TO
*                                          10 WORDS
*-- THE FOLLOWING MAY BE CHANGED FOR OTHER REQUESTS --*
*
RM       EQU      *                  REQUEST
RMHBSCC  DATA     X'1002'              BSC CTRL CHARS (DLE STX)
RMHID    DATA     C'X'                 HEADER ID
RMHTYP   DATA     C'R'                 HEADER TYPE: REQUEST
RMREQ    DATA     F'2'                 REQUEST TYPE: ALLOCATE
RMADSN   DATA     CL8'MYDATA'          DATA SET NAME: MYDATA
RMAVOL   DATA     CL6'MYVOL'           VOLUME NAME: MYVOL
RMANREC  DATA     D'10'                NUMBER RECORDS: 10
RMADST   DATA     F'1'                 DATA SET TYPE: DATA
REQLEN   EQU      *-RM               LENGTH OF REQUEST
*
         ENDPROG
         END
```

Figure 53 (Part 2 of 2). $RMU Multifunction Program

Sample Programs *(continued)*

RECEIVE Sample Program

This sample program, which runs on the host, sends a RECEIVE request transferring a data set to the remote Series/1. The blocking factor for the data is 2, and it is transferred in 80-byte records.

```
EXRECV   PROGRAM START,DS=((RECVDS,??))
START    EQU   *
         BSCOPEN  IOCB,ERROR=BSCOPEN    OPEN BSC LINE
*
         MOVE     IOCB3,+REQLEN         LENGTH OF REQUEST IN IOCB
         BSCWRITE IX,IOCB,ERROR=BSCERR  WRITE REQUEST
         BSCWRITE E,IOCB,ERROR=BSCERR   WRITE EOT
*
         MOVEA    IOCB2,ST              ADDRESS OF STATUS
         MOVE     IOCB3,+STL            LENGTH OF STATUS IN IOCB
         BSCREAD  I,IOCB,ERROR=BSCERR   READ STATUS
         BSCREAD  C,IOCB,ERROR=BSCERR   READ EOT
         IF       (STSFN,NE,-1)         IF STATUS INDICATES ERROR
           PRINTEXT '@STATUS INDICATES ERROR'  THEN PRINT IT
           PRINTNUM ST,5,MODE=HEX
           GOTO     TERM1                  TERMINATE
         ENDIF                           ENDIF
*
         MOVEA    IOCB2,DT              ADDRESS OF DATA
         MOVE     IOCB3,+DTL            SET LENGTH
DATA     EQU      *
         READ     DS1,DISKREC,ERROR=RDERR,END=RDEND  READ RECORD
         MOVE     DTDATA,DISKREC,(80,BYTE)         FIRST RECORD
         MOVE     DTDATA+80,DISKREC+128,(80,BYTE) SECOND RECORD
         IF       (COUNT,EQ,0)          IF FIRST TIME THEN
           BSCWRITE IX,IOCB,ERROR=BSCERR,END=BSCAB WRITE INITIAL
         ELSE                           ELSE
           BSCWRITE CX,IOCB,ERROR=BSCERR,END=BSCAB WRITE CONTINUE
         ENDIF                          ENDIF
         ADD      COUNT,2               ADD 2 TO COUNT
         GOTO     DATA                  CONTINUE TRANSFERRING DATA
RDEND    EQU      *                     TO HERE WHEN AT ENDFILE
         BSCWRITE E,IOCB,ERROR=BSCERR   WRITE EOT
         BSCREAD  I,IOCB,ERROR=BSCERR   READ COUNT
         BSCREAD  C,IOCB,ERROR=BSCERR   READ EOT
         IF       (DTCCNT,EQ,COUNT)     IF COUNT OK THEN
           PRINTEXT 'COUNT OK:'            PRINT IT
           PRINTNUM COUNT
         ELSE                           ELSE
           PRINTEXT '@COUNT FAILED.  COUNTED:'
           PRINTNUM COUNT                  PRINT COUNTS
           PRINTEXT '  COUNT RECORD:'
           PRINTNUM DTCCNT
         ENDIF                          ENDIF
```

Figure 54 (Part 1 of 4). RECEIVE Sample Program

```
TERM1    EQU      *                EXIT POINT FOR NORMAL TERM
         BSCCLOSE IOCB             CLOSE BSC LINE
TERM2    EQU      *                EXIT POINT FOR OPEN FAILED
         PROGSTOP
BSCAB    EQU      *                    ABORT RECEIVED ON WRITE
         BSCREAD  I,IOCB,ERROR=BSCERR  READ STATUS
         BSCREAD  C,IOCB,ERROR=BSCERR  READ EOT
         PRINTEXT '@ABORT RECEIVED.  STATUS:'
         PRINTNUM DT,5,MODE=HEX
         GOTO     TERM1                TERMINATE
*
BSCERR   EQU   *              BSC ERROR ROUTINE
         MOVE     ST,EXRECV           MOVE RETURN CODE
         PRINTEXT '@BSC ERROR:'
         PRINTNUM ST                  PRINT RETURN CODE
         GOTO     TERM1               GO TO TERMINATION
*
BSCOPEN  EQU   *               OPEN ERROR
         MOVE     ST,EXRECV           MOVE RETURN CODE
         PRINTEXT '@BSC OPEN ERROR:'
         PRINTNUM ST                  PRINT RETURN CODE
         GOTO     TERM2               GO TO TERMINATION
```

Figure 54 (Part 2 of 4). RECEIVE Sample Program

```
*
RDERR    EQU   *                DISK READ ERROR
         MOVE     ST,EXRECV           MOVE RETURN CODE
         PRINTEXT '@DISK READ ERROR:'
         PRINTNUM ST                  PRINT RETURN CODE
         MOVEA    IOCB2,ST            POINT IOCB TO
*                                      STATUS MESSAGE
         MOVE     IOCB3,4             SET LENGTH TO 4
         MOVE     ST,X'1002'          SET UP STATUS MESSAGE
         MOVE     ST+2,C'XS'
         BSCWRITE IX,IOCB,ERROR=BSCERR SEND STATUS MESSAGE
         BSCWRITE E,IOCB,ERROR=BSCERR  SEND EOT
         GOTO     TERM2               GO TO TERMINATION
*
IOCB     BSCIOCB  9,RM,0,P2=IOCB2,P3=IOCB3      IOCB
*                                 P2=  IS RECORD ADDRESS
*                                 P3=  IS RECORD LENGTH
*
RLEN     DATA     F'0'           RECORD LENGTH
*
COUNT    DATA     F'0'           RECORD COUNT
*--  REQUEST FOR $RMU TO RECEIVE A DATA SET
*
RM       EQU   *                 REQUEST
RMHBSCC  DATA  X'1002'           BSC CNTRL CHARS (DLE STX)
RMHID    DATA  C'X'              HEADER ID
RMHTYP   DATA  C'R'              HEADER TYPE:     REQUEST
RMREQ    DATA  F'1'              REQUEST TYPE:    RECEIVE
RMRDSN   DATA  CL8'MYDATA'       DATA SET NAME:   MYDATA
RMRVOL   DATA  CL6'      '       VOLUME NAME:     (IPL VOL)
RMRSTR   DATA  D'0'              STARTING RECORD: NONE
RMRTYP   DATA  F'1'              RECEIVE TYPE:    SOURCE
RMRBLK   DATA  F'2'              BLOCKING FACTOR: 2
REQLEN   EQU   *-RM              LENGTH OF REQUEST
```

Figure 54 (Part 3 of 4). RECEIVE Sample Program

```
*--  STATUS RECORD
*
ST        DATA  10F'0'            AREA FOR STATUS RECORD
*
*
*
STSFN     EQU   ST+6              STATUS FUNCTION
STL       EQU   *-ST              STATUS RECORD LENGTH
*
*--  DATA AND COUNT RECORD
*
DT        DATA  X'1002'           DATA RECORD:  DLE STX
          DATA  C'XD'             HEADER ID, TYPE (DATA)
DTCCNT    EQU   DT+10             LOCATION OF COUNT
DTDATA    DATA  160C' '
DTL       EQU   *-DT              LENGTH
*
DISKREC   DATA  128F'0'           DISK RECORD AREA
          ENDPROG
          END
```

Figure 54 (Part 4 of 4). RECEIVE Sample Program

SEND Sample Program

This program, which runs on the host, contains a SEND request. It asks the remote system to transfer a data set to the host. Data is blocked with a factor of 3, and transferred in 256-byte records.

```
EXSEND   PROGRAM START,DS=((SENDDS,??))
START    EQU   *
         BSCOPEN  IOCB,ERROR=BSCOPEN    OPEN BSC LINE
         MOVE     IOCB3,+REQLEN         LENGTH OF REQUEST IN IOCB
         BSCWRITE IX,IOCB,ERROR=BSCERR  WRITE REQUEST
         BSCWRITE E,IOCB,ERROR=BSCERR   WRITE EOT
         MOVEA    IOCB2,ST              ADDRESS OF STATUS
         MOVE     IOCB3,+STL            LENGTH OF STATUS IN IOCB
         BSCREAD  I,IOCB,ERROR=BSCERR   READ STATUS
         IF       (STSFN,NE,-1)         IF STATUS INDICATES ERROR
           BSCREAD  C,IOCB,ERROR=BSCERR   READ EOT
           PRINTEXT '@STATUS INDICATES ERROR'  THEN PRINT IT
           PRINTNUM ST,5,MODE=HEX
           GOTO     TERM1                   TERMINATE
         ENDIF                           ENDIF
         MOVEA    IOCB2,DT              ADDRESS OF DATA
DATA     EQU      *
         MOVE     IOCB3,+DTL            SET LENGTH TO MAX
         BSCREAD  C,IOCB,ERROR=BSCERR   READ DATA OR COUNT
         SUB      IOCB,IOCB2,RESULT=RLEN  COMPUTE LENGTH
         IF       (DTHTYPR,EQ,C'D',BYTE) IF DATA THEN
           SUB      RLEN,+4               -4 FROM LENGTH
*                                         FOR HEADER
           SHIFTR   RLEN,8                RLEN = NUMBER RECORDS
*                                         WRITE RECORDS NEXT
           WRITE    DS1,DTDATA,RLEN,ERROR=WRERR,END=WRERR
           ADD      COUNT,RLEN            ADD NUMBER WRITTEN
*                                         TO COUNT
           GOTO     DATA                  GO READ NEXT RECORD
         ELSE                            ELSE
           IF       (DTHTYPR,EQ,C'C',BYTE)  IF COUNT THEN
             IF       (DTCCNT,EQ,COUNT)     IF COUNT OK THEN
               PRINTEXT 'COUNT OK:'           PRINT IT
               PRINTNUM COUNT
             ELSE                             ELSE
               PRINTEXT 'COUNT FAILED.  COUNTED:'
               PRINTNUM COUNT                   PRINT COUNTS
               PRINTEXT '  COUNT RECORD:'
               PRINTNUM DTCCNT
             ENDIF                            ENDIF
           ELSE                             ELSE MUST BE STATUS
             PRINTEXT 'ERROR MSG RECEIVED:'
             PRINTNUM DT,5,MODE=HEX           PRINT IT
           ENDIF                            ENDIF
         ENDIF                           ENDIF
         BSCREAD  C,IOCB,ERROR=BSCERR   READ EOT
```

Figure 55 (Part 1 of 3). SEND Sample Program

```
TERM1    EQU      *                EXIT POINT FOR NORMAL TERM
         BSCCLOSE IOCB             CLOSE BSC LINE
TERM2    EQU      *                EXIT POINT FOR OPEN FAILED
         PROGSTOP
BSCERR   EQU   *                BSC ERROR ROUTINE
         MOVE     ST,EXSEND           MOVE RETURN CODE
         PRINTEXT '@BSC ERROR:'
         PRINTNUM ST                  PRINT RETURN CODE
         GOTO     TERM1               GO TO TERMINATION
*
BSCOPEN  EQU   *                OPEN ERROR
         MOVE     ST,EXSEND           MOVE RETURN CODE
         PRINTEXT '@BSC OPEN ERROR:'
         PRINTNUM ST                  PRINT RETURN CODE
         GOTO     TERM2               GO TO TERMINATION
*
WRERR    EQU   *                WRITE ERROR
         MOVE     ST,EXSEND           MOVE RETURN CODE
         PRINTEXT '@DISK WRITE ERROR:'
         PRINTNUM ST                  PRINT RETURN CODE
         BSCWRITE E,IOCB,ERROR=BSCERR WRITE EOT (ABORT)
         MOVEA    IOCB2,ST            POINT IOCB TO STATUS
         MOVE     IOCB3,4             SET LENGTH TO 4
         MOVE     ST,X'1002'          SET UP STATUS MESSAGE
         MOVE     ST+2,C'XS'
         BSCWRITE IX,IOCB,ERROR=BSCERR WRITE STATUS
         BSCWRITE E,IOCB,ERROR=BSCERR  WRITE EOT
         GOTO     TERM1               GO TO TERMINATION
```

Figure 55 (Part 2 of 3). SEND Sample Program

```
*
IOCB     BSCIOCB   9,RM,0,P2=IOCB2,P3=IOCB3     IOCB
*
*                              P2=IOCB2 IDENTIFIES MSG ADDRESS
*                              P3=IOCB3 IDENTIFIES MSG LENGTH
RLEN     DATA      F'0'         RECORD LENGTH
*
COUNT    DATA      F'0'         RECORD COUNT
*
*--  REQUEST FOR $RMU TO SEND DATA SET
*
RM       EQU   *                REQUEST
RMHBSCC  DATA  X'1002'            BSC CNTRL CHARS (DLE STX)
RMHID    DATA  C'X'               HEADER ID
RMHTYP   DATA  C'R'               HEADER TYPE:     REQUEST
RMREQ    DATA  F'0'               REQUEST TYPE:    SEND
RMSDSN   DATA  CL8'MYDATA'        DATA SET NAME:   MYDATA
RMSVOL   DATA  CL6'      '        VOLUME NAME:     (IPL VOL)
RMSSTR   DATA  D'0'               STARTING RECORD: NONE
RMSTYP   DATA  F'0'               SEND TYPE:       NORMAL
RMSBLK   DATA  F'3'               BLOCKING FACTOR: 3
REQLEN   EQU   *-RM              LENGTH OF REQUEST
*--  STATUS RECORD
*
ST       DATA  10F'0'             AREA FOR STATUS RECORD
*
*
*
STSFN    EQU   ST+6              STATUS FUNCTION
STL      EQU   *-ST              STATUS RECORD LENGTH
*
*--  DATA AND COUNT RECORD
*
DT       DATA  387F'0'           AREA FOR DATA RECORD
*                                 4 BYTES  MESSAGE HEADER
*                                768 BYTES  3 256-BYTE RECS
*                                 1 BYTE   ETX
*                                ---------
*                                773 BYTES  TOTAL, ROUNDED UP
*                                           TO 387 WORDS
DTHTYPR  EQU   DT+3              RECORD TYPE
DTDATA   EQU   DT+4              DATA
DTCCNT   EQU   DT+10             COUNT
DTL      EQU   *-DT              LENGTH
         ENDPROG
         END
```

Figure 55 (Part 3 of 3). SEND Sample Program

PASSTHRU Sample Program

This sample program executes a PASSTHRU session. The host Series/1 establishes a session with the supervisor of a remote Series/1. The program uses blocking. The host terminal looks as if it were connected to the remote system.

```
EXPASST  PROGRAM START,TERMERR=TERM1
*
*        THIS EXAMPLE HOST PROGRAM USES THE PASSTHRU FUNCTION
*        OF THE REMOTE MANAGEMENT UTILITY. THE OPERATOR IS
*        ASKED WHETHER TO START THE PASSTHRU ASSIST PROGRAM.
*        IF SO, THE PROGRAM $RMUPA IS INVOKED. AFTER THIS, A
*        SESSION IS ESTABLISHED WITH THE EDX SUPERVISOR.
*
*        WHENEVER A "PROGRAM END" PASSTHRU RECORD IS RECEIVED,
*        A "REQUEST DATA" RECORD IS SENT.  WHEN A "NO DATA"
*        RECORD IS RECEIVED, THE OPERATOR IS ASKED WHETHER TO
*        "ATTN" (END THE SESSION AND START ANOTHER), "READ"
*        (TRY TO ACQUIRE DATA FROM THE HOST), OR "QUIT" (END
*        THE PASSTHRU SESSION AND THEN TERMINATE.
*
START    EQU   *
         BSCOPEN  IOCB,ERROR=BSCOPEN   OPEN BSC LINE
*
*--  START UP PASSTHRU ASSIST PROGRAM ($RMUPA) IF NEEDED
*
         QUESTION 'START PASSTHRU ASSIST PROGRAM?',NO=START2
*
         MOVEA    IOCB2,REQPTAS        ADDRESS OF REQUEST IN IOCB
         MOVE     IOCB3,+REQPTASL      LENGTH OF REQUEST IN IOCB
         BSCWRITE IX,IOCB,ERROR=BSCERR WRITE REQUEST
         BSCWRITE E,IOCB,ERROR=BSCERR  WRITE EOT
*
         MOVEA    IOCB2,ST             ADDRESS OF STATUS
         MOVE     IOCB3,+STL           LENGTH OF STATUS IN IOCB
         BSCREAD  I,IOCB,ERROR=BSCERR  READ STATUS
         BSCREAD  C,IOCB,ERROR=BSCERR  READ EOT
         IF       (STSFN,NE,-1)        IF STATUS INDICATES ERROR
           PRINTEXT '@STATUS INDICATES ERROR'  PRINT IT
           PRINTNUM ST,5,MODE=HEX
           GOTO     TERM1                TERMINATE
         ENDIF                         ENDIF
         MOVEA    IOCB2,DT             ADDRESS OF DATA
         MOVE     IOCB3,+DTL           SET LENGTH
         BSCREAD  I,IOCB,ERROR=BSCERR,TIMEOUT=NO
*                                      READ, EXPECT PROGRAM END
         BSCREAD  C,IOCB,ERROR=BSCERR,TIMEOUT=NO    READ EOT
         IF       (EXPASST,EQ,+1),AND,(DT+RMPTYP,EQ,+RMPTYPPE)
*                                      IF PGM END AND EOT THEN
           MOVE     DT,X'1002'           SET UP PTHRU PGM END
           MOVE     DT+RMPTYP,+RMPTYPPE  PTHRU TYPE IS PGM END
           MOVE     IOCB3,+RMPX          SET UP LENGTH IN IOCB
           BSCWRITE IX,IOCB,ERROR=BSCERR,END=BSCAB WRITE TO RMU
           BSCWRITE E,IOCB,ERROR=BSCERR  WRITE EOT
```

Figure 56 (Part 1 of 7). PASSTHRU Sample Program

```
         ELSE                              ELSE
           MOVE     ST,EXPASST              SAVE RETURN CODE
           PRINTEXT '@UNSUCCESSFUL LOAD OF PASSTHRU ASSIST PGM.'
           PRINTEXT '@LAST MESSAGE READ:'
           PRINTNUM DT,10,MODE=HEX          PRINT MESSAGE
           PRINTEXT '@LAST RETURN CODE FROM READ:'
           PRINTNUM ST,MODE=HEX             PRINT RETURN CODE
           GOTO     TERM1                   TERMINATE
         ENDIF                             ENDIF
*
*--  MAIN PASSTHRU PROCESSING.  SEND REQUEST
*
START2   MOVEA    IOCB2,REQPT              ADDRESS OF REQUEST IN IOCB
         MOVE     IOCB3,+REQLEN            LENGTH OF REQUEST IN IOCB
         BSCWRITE IX,IOCB,ERROR=BSCERR     WRITE REQUEST
         BSCWRITE E,IOCB,ERROR=BSCERR      WRITE EOT
*
         MOVEA    IOCB2,ST                 ADDRESS OF STATUS
         MOVE     IOCB3,+STL               LENGTH OF STATUS IN IOCB
         BSCREAD  I,IOCB,ERROR=BSCERR      READ STATUS
         BSCREAD  C,IOCB,ERROR=BSCERR      READ EOT
         IF       (STSFN,NE,-1)            IF STATUS INDICATES ERROR
           PRINTEXT '@STATUS INDICATES ERROR'  PRINT IT
           PRINTNUM ST,5,MODE=HEX
           GOTO     TERM1                    TERMINATE
         ENDIF                             ENDIF
```

Figure 56 (Part 2 of 7). PASSTHRU Sample Program

```
READ     EQU      *
         MOVEA    IOCB2,DT              ADDRESS OF DATA
         MOVE     IOCB3,+DTL            SET LENGTH
         IF       (BSCST,NE,+BSCSTRD)   IF BSC STATE IS NOT READ
           BSCREAD  I,IOCB,ERROR=BSCERR,TIMEOUT=NO     READ INIT
           MOVE   BSCST,+BSCSTRD          BSC STATE = READ
         ELSE                           ELSE
           BSCREAD  C,IOCB,ERROR=BSCERR,TIMEOUT=NO     READ CONT
         ENDIF                          ENDIF
*
         IF       (DT+RMHTYP,NE,C'P',BYTE)  IF NOT PASSTHRU THEN
           PRINTEXT '@NON-PASSTHRU MESSAGE RECEIVED:'
           PRINTNUM DT,5,MODE=HEX         PRINT WHAT WAS RECEIVED
*                                          (WILL BE STATUS)
           BSCREAD  C,IOCB,ERROR=BSCERR,TIMEOUT=NO     READ EOT
           GOTO     TERM1                 TERMINATE
         ENDIF                          ENDIF
*--  CASE:  PASSTHRU TYPE
         GOTO     (ERRPT,TEXT,REQD,PGME,NODA),DT+RMPTYP
*
TEXT     EQU      *                     PASSTHRU TYPE:  DATA
         MOVEA    #1,DT+RMPST           SET #1 TO BEGINNING OF TXT
         DO       UNTIL,(#1,EQ,IOCB)    DO UNTIL AT END OF TEXT
*                                        (IOCB CONTAINS ADDRESS
*                                        OF BYTE PAST LAST BYTE
*                                        OF DATA)
           IF       ((0,#1),EQ,-1),OR,((0,#1),EQ,-2) IF TEXT
             PRINTEXT (4,#1),MODE=LINE     PRINT TO TERMINAL
             IF       ((0,#1),EQ,-1)       IF NEWLINE
               PRINTEXT SKIP=1               THEN DO NEWLINE
             ENDIF                          ENDIF
             ADD      #1,(2,#1)             POINT #1 TO NEXT TEXT
             ADD      #1,5                  ADD HEADER LENGTH + 1
             AND      #1,X'FFFE'            POINT TO EVEN BOUNDARY
           ELSE                           ELSE
             IF       ((0,#1),EQ,X'8F',BYTE) IF LINE= THEN
               AND      (0,#1),X'00FF',RESULT=N1  DO IT
               PRINTEXT LINE=N1               ON TERMINAL
             ELSE                           ELSE
               IF       ((0,#1),EQ,X'8E',BYTE) IF SKIP= THEN
                 AND      (0,#1),X'00FF',RESULT=N1  DO IT
                 PRINTEXT SKIP=N1               ON TERMINAL
               ENDIF                          ENDIF
             ENDIF                          ENDIF
             ADD      #1,4                  POINT #1 TO NEXT
*                                          TEXT BLOCK
           ENDIF                          ENDIF
         ENDDO                          ENDDO
         GOTO     READ                  END TEXT PROCESSING
```

Figure 56 (Part 3 of 7). PASSTHRU Sample Program

```
REQD     EQU      *                       PASSTHRU TYPE:  REQ DATA
         BSCREAD  C,IOCB,ERROR=BSCERR     READ EOT
         MOVE     DT+RMPTXTL,X'FE00'      SET UP "TEXT" STATEMENT
         READTEXT DT+RMPTXT,MODE=LINE     GET TEXT FROM TERMINAL
         MOVE     DT,X'1002'              SET UP PTHRU TEXT RECORD
         MOVE     DT+RMPTYP,+RMPTYPTX     PTHRU TYPE IS TEXT OR PFK
         MOVE     DT+RMPTXTL,0,BYTE       ZERO HI-ORDER LENGTH BYTE
         IF       (DT+RMPTXTL,GE,4),AND,(DT+RMPTXT,EQ,C'$P'),
                AND,(DT+TXT2,EQ,C'F',BYTE)  IF "$PFN" ENTERED
           MOVE     DT+RMPTXTL,-1           INDICATE PF KEY
           MOVE     DT+RMPTXT,DT+TXT2       PLACE NUMBER IN MSG
           AND      DT+RMPTXT,X'000F'       PURIFY NUMBER
           MOVE     IOCB3,2+RMPTXT          LENGTH IN IOCB
         ELSE                               ELSE
           MOVE     IOCB3,DT+RMPTXTL        SET UP LENGTH IN IOCB
           ADD      IOCB3,+RMPTXT           INCLUDING HEADER
         ENDIF                              ENDIF
         BSCWRITE IX,IOCB,ERROR=BSCERR,END=BSCAB WRITE TO RMU
         BSCWRITE E,IOCB,ERROR=BSCERR     WRITE EOT
         MOVE     BSCST,+BSCSTO           BSC STATE = RESET
         GOTO     READ                    END REQ TEXT PROCESSING
*
PGME     EQU      *                  PASSTHRU TYPE:  PROGRAM END
*                                                     (DISCONNECT)
         BSCREAD  C,IOCB,ERROR=BSCERR     READ EOT
         GOTO     SNDRQD                  GO AND REQUEST DATA
*
NODA     EQU      *                       PASSTHRU TYPE:  NO DATA
         BSCREAD  C,IOCB,ERROR=BSCERR     READ EOT
NODAQ    PRINTEXT '@"NO DATA" RECEIVED.  ENTER ONE:'
         READTEXT INMSG,'@  A(TTN), R(EAD), Q(UIT) '
         IF       (INMSG,EQ,C'A',BYTE),OR,(INMSG,EQ,C'Q',BYTE)
*                                        IF "ATTN" OR "QUIT" THEN
*                                          SEND PROGRAM END
           MOVE     DT,X'1002'              SET UP PTHRU PGM END
           MOVE     DT+RMPTYP,+RMPTYPPE     PTHRU TYPE IS PGM END
           MOVE     IOCB3,+RMPX             SET UP LENGTH IN IOCB
           BSCWRITE IX,IOCB,ERROR=BSCERR,END=BSCAB WRITE TO RMU
           BSCWRITE E,IOCB,ERROR=BSCERR     WRITE EOT
           MOVE     BSCST,+BSCSTO           BSC STATE = RESET
```

Figure 56 (Part 4 of 7). PASSTHRU Sample Program

```
          IF       (INMSG,EQ,C'A',BYTE),GOTO,START2
*                                        IF "A" THEN START NEW
*                                        SESSION
          GOTO     TERM1                 OTHERWISE TERMINATE
        ELSE                           ELSE (NOT "ATTN"
*                                             OR "QUIT")
          IF       (INMSG,EQ,C'R'),GOTO,SNDRQD  IF "R" THEN
*                                               REQUEST DATA
          GOTO     NODAQ                 ELSE ASK AGAIN
        ENDIF                          ENDIF
ERRPT   EQU      *                     PASSTHRU TYPE:  UNKNOWN
        PRINTEXT '@INVALID PASSTHRU RECORD RECEIVED:'
        PRINTNUM DT,20,MODE=HEX
        GOTO     TERM1                 TERMINATE
*
*--     END OF CASES
*
SNDRQD  EQU      *                     SEND REQUEST DATA
        MOVE     DT,X'1002'            SET UP PTHRU REQUEST DATA
        MOVE     DT+RMPTYP,+RMPTYPRD   PTHRU TYPE IS REQEST DATA
        MOVE     IOCB3,+RMPX           SET UP LENGTH IN IOCB
        BSCWRITE IX,IOCB,ERROR=BSCERR,END=BSCAB WRITE TO RMU
        BSCWRITE E,IOCB,ERROR=BSCERR   WRITE EOT
        MOVE     BSCST,+BSCSTO         BSC STATE = RESET
        GOTO     READ                  END REQ TEXT PROCESSING
*
*
TERM1   EQU      *                EXIT POINT FOR NORMAL TERM
        BSCCLOSE IOCB             CLOSE BSC LINE
TERM2   EQU      *                EXIT POINT FOR OPEN FAILED
        PROGSTOP
*
BSCAB   EQU      *                     ABORT RECEIVED ON WRITE
        BSCREAD  I,IOCB,ERROR=BSCERR   READ STATUS
        BSCREAD  C,IOCB,ERROR=BSCERR   READ EOT
        PRINTEXT '@ABORT RECEIVED.  STATUS:'
        PRINTNUM DT,20,MODE=HEX
        GOTO     TERM1                 TERMINATE
```

Figure 56 (Part 5 of 7). PASSTHRU Sample Program

```
*
BSCERR   EQU   *                BSC ERROR ROUTINE
         MOVE     ST,EXPASST           MOVE RETURN CODE
         PRINTEXT '@BSC ERROR:'
         PRINTNUM ST                   PRINT RETURN CODE
         GOTO     TERM1                GO TO TERMINATION
*
BSCOPEN  EQU   *                OPEN ERROR
         MOVE     ST,EXPASST           MOVE RETURN CODE
         PRINTEXT '@BSC OPEN ERROR:'
         PRINTNUM ST                   PRINT RETURN CODE
         GOTO     TERM2                GO TO TERMINATION
*--  DATA AREA
*
INMSG    TEXT     LENGTH=4          INPUT MSG FROM OPERATOR
*
IOCB     BSCIOCB  9,0,0,P2=IOCB2,P3=IOCB3     IOCB
*                                  P2= IS RECORD ADDRESS
*                                  P3= IS RECORD LENGTH
*
*--  REQUEST FOR PASSTHRU
*
REQPT    EQU   *                REQUEST
         DATA  X'1002'            BSC CONTROL CHARS (DLE STX)
         DATA  C'X'               HEADER ID
         DATA  C'R'               HEADER TYPE:     REQUEST
         DATA  A(RMREQPST)        REQUEST TYPE:    PASSTHRU (12)
         DATA  A(PBL)             PASSTHRU BLKING
         DATA  H'0'               FLAG (UNUSED)
         DATA  H'0'               PARTITION (UNUSED)
         DATA  CL8' '             PROGRAM:      EDX SUPERVISOR
         DATA  CL6' '             VOLUME (UNUSED)
         DATA  3F'0'              (REMAINDER UNUSED)
REQLEN   EQU   *-REQPT          LENGTH OF REQUEST
```

Figure 56 (Part 6 of 7). PASSTHRU Sample Program

```
*
*--  PASSTHRU REQUEST:  START PASSTHRU ASSIST PROGRAM
*
REQPTAS  EQU   *                 REQUEST
         DATA  X'1002'             BSC CONTROL CHARS (DLE STX)
         DATA  C'X'                HEADER ID
         DATA  C'R'                HEADER TYPE:      REQUEST
         DATA  A(RMREQPST)         REQUEST TYPE:     PASSTHRU (12)
         DATA  A(0)                PASSTHRU BLKING   (NONE)
         DATA  H'0'                FLAG (UNUSED)
         DATA  H'0'                PARTITION         (ANY)
         DATA  CL8'$RMUPA'         PROGRAM:          $RMUPA
         DATA  CL6' '              VOLUME:           IPL
         DATA  F'0'                FREE SPACE:       NONE
         DATA  F'0'                PARAMETERS:       NONE
         DATA  F'0'                DATA SETS:        NONE
REQPTASL EQU   *-REQPTAS         LENGTH OF REQUEST
*
*--  STATUS RECORD
*
ST       DATA  10F'0'             AREA FOR STATUS RECORD
*
*
*
STSFN    EQU   ST+6              STATUS FUNCTION
STL      EQU   *-ST              STATUS RECORD LENGTH
*
*--  PASSTHRU SESSION AREA
*
DT       DATA  256F'0'           RECORD
DTL      EQU   *-DT              LENGTH
PBL      EQU   DTL-8             PASSTHRU BLOCK LENGTH
*                                  LENGTH OF DATA AREA -
*                                  6 BYTES FOR HEADER AND 2
*                                  FOR ETX AND WORD ROUND UP
*
*--  MISCELLANEOUS VARIABLES
*
BSCST    DATA  F'0'              BSC STATE:
BSCSTO   EQU   0                   RESET
BSCSTRD  EQU   1                   READING
N1       DATA  F'0'              WORK WORD
*
         COPY  CDRRM             INCLUDE DEFINITION OF RMU MSGS
TXT2     EQU   RMPTXT+2          BYTE 2 OF PASSTHRU TEXT
*
         ENDPROG
         END
```

Figure 56 (Part 7 of 7). PASSTHRU Sample Program

Sample Programs *(continued)*

Example of Conducting a PASSTHRU Session

In this example of conducting a PASSTHRU session, the host invokes and runs the $DEBUG utility.

```
> $L EXPASST
EXPASST 9P LP=C900

START PASSTHRU ASSIST PROGRAM?  Y

> $L $DEBUG
$DEBUG  27P,09:44:08  LP=BF00

PROGRAM NAME:  $DISKUT1
$DISKUT1  30P,09:44:14 LP=DA00

REQUEST ¶¶HELP◀◀ TO GET LIST OF DEBUG COMMANDS
TASK   STOPPED AT 0064
¶¶NO DATA◀◀ RECEIVED.  ENTER ONE:
  A(TTN), R(EAD), Q(UIT)  A
> WHERE
TASK   STOPPED AT 0064
$ATTASK AT 2600
¶¶NO DATA◀◀ RECEIVED.  ENTER ONE:
  A(TTN), R(EAD), Q(UIT)  A
```

Figure 57 (Part 1 of 4). Example of Conducting a PASSTHRU Session

```
> GO
OPTION(*/ADDR/TASK/ALL):  ALL
  1 BREAKPOINT(S) ACTIVATED
USING VOLUME EDX002

COMMAND (?):  LA ZZZZ
USING VOLUME EDX002
  NAME      FREC  SIZE
  12845 FREE RECORDS IN LIBRARY
COMMAND (?):  $PFO
XX
> WHERE
INVALID COMMAND
TASK     AT  0274
$ATTASK  AT 2600
```

Figure 57 (Part 2 of 4). Example of Conducting a PASSTHRU Session

```
COMMAND (?):  $PFO
XX
> AT
INVALID COMMAND
OPTION(*/ADDR/TASK/ALL):  A
BREAKPOINT ADDR:  274
LIST/NOLIST:  N
STOP/NOSTOP:  S
  1 BREAKPOINT(S) SET
COMMAND:  XX
TASK STOPPED AT 0274
¶¶NO DATA◀◀ RECEIVED.  ENTER ONE:
  A(TTN), R(EAD), Q(UIT)  A
> LIST A 274 5 X
0274 X' 80AF 1010 C9D5 E5C1 D3C9'
¶¶NO DATA◀◀ RECEIVED.  ENTER ONE:
  A(TTN), R(EAD), Q(UIT)  A
> END
  1 BREAKPOINT(S) REMOVED
INVALID COMMAND
```

Figure 57 (Part 3 of 4). Example of Conducting a PASSTHRU Session

```
COMMAND (?):  EN
¶¶NO DATA◀◀ RECEIVED.  ENTER ONE:
  A(TTN), R(EAD), Q(UIT)  R
¶¶NO DATA◀◀ RECEIVED.  ENTER ONE:
  A(TTN), R(EAD), Q(UIT)  A
> $RMUPA
¶¶NO DATA◀◀ RECEIVED.  ENTER ONE:
  A(TTN), R(EAD), Q(UIT)  A
> $A
PROGRAMS AT 09:50:26
IN PARTITION #1   NONE
¶¶NO DATA◀◀ RECEIVED.  ENTER ONE:
  A(TTN), R(EAD), Q(UIT)  Q
EXPASST ENDED
```

Figure 57 (Part 4 of 4). Example of Conducting a PASSTHRU Session

Chapter 3. Host Communications Facility

When Series/1 has the Host Communication Facility (HCF) loaded on it, it can communicate with a host system to perform various functions. The host system has the Host Communication Facility Installed User Program (IUP 5796-PGH) executing on it. The Host Communications Facility allows the Series/1 to perform file transfers and to submit job streams to the host.

You must write the application program that communicates with the host system. It must contain Event Driven Language TP instructions. These instructions perform various communications functions between the Series/1 and the host. Your program can perform the following functions:

- Write to a host data set
- Read from a host data set
- Submit a background job to the host system
- Obtain the time and date from the host system
- Set the occurrence of a Series/1 event so that it can be tested by a program running on the host system
- Test for the occurrence of an event that is set by the host system
- Erase the record, on the host system, of an event that occurred on either the Series/1 or the host system.

You can also perform HCF functions with the $HCFUT1 utility, which provides interactive capability between the Series/1 and the host.

Planning to Use the Host Communications Facility

Certain requirements and restrictions apply to the operation of the Host Communications Facility.

Installation Requirements

Both the host and Series/1 must meet certain installation requirements for successful communications through HCF. The host must have the HCF IUP 5796-PGH executing on it. The BSC line connecting the Series/1 to the System/370 must be point-to-point leased. Only the BSC Single Line Control (feature #2074) can be used to attach the line to the Series/1.

System generation for the Series/1 must support the Host Communications Facility. The appropriate supervisor modules and the TPCOMM system configuration statement provide this support. Refer to the *Installation and System Generation Guide* for information.

Host Data Sets

Host data sets in your HCF programs must be named according to a naming convention. Also, your program can open only one host data set at a time.

Host Data Set Naming Conventions

When you refer to a host data set in a TP instruction, its name must consist of an alphanumeric character string immediately preceded by one word specifying the length of the name field. You can do this most easily by coding a labeled TEXT instruction to define the name; for example:

```
DSN1    TEXT 'XYZ.EXP1.DATA
```

Data set names follow standard host system naming conventions and must not exceed 44 characters in length (including delimiting periods). Pad the name field with blanks on the right.

In the case of a partitioned data set and member name, specify a string of the form dsname(membername); for example:

```
PDSDSN  TEXT 'XYZ.EXP1.DATA(RUN1)
```

The maximum length of such a string is 54 characters.

To read a data set name from a terminal into a text field, issue a READTEXT instruction.

Host Data Set Characteristics

You can access host data sets with the following characteristics:

- they must be catalogued
- they must be single volume
- they must be direct-access
- they must contain fixed or variable-length records
- they can be either sequential data sets or members of partitioned data sets.
- they can be either blocked or unblocked.

Fixed-length logical records must contain an even number of words. In fixed blocked format the block size must be an integer multiple of the logical record length (LRECL), not exceeding 13030.

You can use either sequential data sets or members of partitioned data sets to submit jobs to the host. Logical records must be 80 bytes long and can be blocked or unblocked. The size of blocked records must be a multiple of 80.

Record Sizes

You can use a large range of logical and physical record sizes when writing your program. In selecting a record size, there is no absolute best choice, but you should consider the following:

- The basic disk or diskette record size on the Series/1 is 256 bytes. Therefore, this is a natural unit of measure for transfer to and from disk and a natural choice for a logical record size on the host. This is the default for the TP READ and TP WRITE instructions.
- A host physical record (block) size of 1536 bytes yields 80 percent utilization of host direct access storage on an IBM 3330 disk. This size also provides enough record space so that there will be moderate requirements for buffer storage.
- The larger the physical record being transferred between the host and the Series/1 (a host logical record), the higher the effective data transfer rate that will be achieved. Also, the larger the physical record (block) being transferred between host processor storage and direct access, the higher will be the effective data rate. The maximum data rate is achieved when using track size records (13030 bytes for the IBM 3330 disk) for both operations.
- The large physical records naturally require correspondingly large buffers in your program. In order to achieve overlapped I/O, multiple buffers are required.

Planning to Use the Host Communications Facility *(continued)*

Variable-Length Records

A variable-length record is always prefixed by four bytes of control information. This is called a Record Descriptor Word or RDW. The RDW consists of two fields.

The length (LL) field (bytes 1 and 2) describes the total length of the record in bytes and is therefore always four greater than the length of the data field. The 00 field (bytes 3 and 4) is reserved for use by the host system.

The rest of the record is taken up by the DATA field.

When a variable-length record is transferred from the host to the Series/1, the total record, including the LL field, is transferred. When a variable-length record is to be transferred from the Series/1 to the host, you must set the RDW to the proper value.

Opening Host Data Sets

You may open only one host data set open at a time. If a second task attempts to open a data set, HCF will place it in a queue of tasks waiting to use the facility.

If the task currently using HCF attempts to open a second data set, then the currently open data set automatically closes, and the second one opens.

System Status Data Set

Synchronization of programs in the host system and the Series/1 is accomplished with a status data set on the host system. Both systems share this data set. You can directly access this system status data set with the TP SET, TP FETCH, and TP RELEASE instructions. With these instructions you set the occurrence of events on the Series/1 which the host monitors, tests for and then erases. These are called status functions. You can also perform the status functions with the $HCFUT1 utility.

For example, one program (Program A) makes an entry in the system status data set by invoking a SET instruction specifying an index and a key. Another program (Program B) tests for the existence of such an entry with a FETCH or RELEASE referring to the same index and key names, and receives a positive return code if the entry exists. After performing a SET, the first program (Program A) could periodically issue a FETCH. A companion program (Program B) on the other system might also be issuing a periodic FETCH for the agreed-upon index and key. At the appropriate time, this program (Program B) could issue a RELEASE which would result in the first program (Program A) receiving a "not found" return code from its next FETCH. This could be interpreted as a notification by the companion program (Program B) that the message had been received.

System Status Data Set Organization

The system status data set has direct organization. You write records into this data set using TP SET, test for the existence of a record using TP FETCH, or test and delete a record using TP RELEASE.

A record sent to or retrieved from the status data set consists of three part, two of which are mandatory:

- Index entry (mandatory)
- Key field (mandatory)
- Data (optional 256-byte field).

Index entries and key fields can each be up to eight EBCDIC characters in length and have significance for the using programs.

The system status data set has one 268-byte index record capable of containing 22 separate index entries. An index entry has two parts:

- Index name - eight EBCDIC characters
- Key pointer - a 4-byte relative record pointer to the first associated key field record.

A key entry is a 268-byte record with the following format:

- Forward pointer - a 4-byte relative record number of the next key entry or zero if this is the last one
- Key name - eight EBCDIC characters
- Data - 256 bytes of optional data.

The next record pointer allows more than one key to be associated with a given index. The next record pointer of the last key field will be set to zero to indicate the end of the chain.

Logically, an unlimited number of key records may be associated with a single index. In practice, the limiting factor is the physical size of the data set. The distributed data set allows for a total of 94 key entries.

The system status data set format is defined and allocated during the installation of the Host Communications Facility Installed User Program.

Appendix B of the *Host Communications Facility Installed User Program*contains more details on the use of the system status data set.

Planning to Use the Host Communications Facility *(continued)*

Host Storage

To ensure economical utilization of host processor storage, while also providing large record capability, host processor storage is shared by all Series/1 systems. The Host Communications Facility IUP region allocation determines how much buffer space is available; therefore, it determines the upper limit for the host BLKSIZE. Despite this determination of buffer size, it is still possible for error code 222 (sufficient I/O buffer space unavailable) to occur because of multiple and simultaneous requests for access to data sets with very large block sizes. Although this is not likely to occur, you should minimize the amount of realtime control you require with the Host Communications Facility in order to minimize the probability of interference.

You should also specifically test for error code 222 in response to a TP OPEN instruction and, if it is received, retry your request later.

Data Transfer Rates

Data transfer rates between a Series/1 and the host vary depending on the activity on the host and the type of physical connection used between the systems. In general, you should avoid implementing any functions in a manner which depends on specific data rates between the host and Series/1.

Tasks Common to Programming and Using $HCFUT1

You can perform almost all tasks both by writing TP instruction programs and using the $HCFUT1 utility. These include transfer of data sets between the host and Series/1, submitting jobs to the host, and performing status functions.

Programming for the Host Communications Facility Application

Your application programs for the Host Communications Facility can control data transfers, submit background jobs to the host, and perform status functions.

Event Driven Language Instruction Set

You write HCF programs with a set of EDL instructions called TP instructions. The chart that follows shows these instructions and the functions they perform. They are listed in alphabetical order.

Instruction	Function
TP CLOSE	Ends a data transfer operation (any operation begun by a TP OPENIN or TP OPENOUT intruction)
TP FETCH	Tests the existence of a record in the system status data set and/or reads it
TP OPENIN	Prepares Series/1 to read data from the host
TP OPENOUT	Prepares the Series/1 to send data to the host
TP READ	Reads data sent from the host to Series/1
TP RELEASE	Deletes a record from system status data set and/or reads it
TP SET	Writes a record to the system status data set
TP SUBMIT	Submits a job stream from the Series/1 to host
TP TIMEDATE	Obtains time of day and date from the host
TP WRITE	Sends data to the host

Figure 58. EDL TP Instructions

For the syntax of the TP instructions, refer to the *Language Reference.*

Controlling Data Transfers between Series/1 and Host

You can send data to the host and receive data from the host through your program.

Sending Data to the Host

Series/1 can write data to a host data set. Code this sequence of TP instructions to perform this function:

1. TP OPENOUT (to specify an output operation)
2. TP WRITE (to send the data to the host system)
3. TP CLOSE (to end the output operation).

In the TP OPENOUT instruction, specify the name of the host data set where you are sending the data. Follow naming conventions.

In the TP WRITE instruction, you must specify the label of the buffer that contains the data to be sent. In the program, specify this buffer with a BUFFER statement that contains the operand TPBSC.

You should also code error routines into the TP WRITE instruction to take over if an error or end condition occurs.

Receiving Data from the Host

Series/1 can ask the host to send it data. Code this sequence of TP instructions to perform this function:

1. TP OPENIN (to specify an input operation)
2. TP READ (to receive the host data)
3. TP CLOSE (to end the receive operation).

In the TP OPENIN instruction, specify the name of the host data set that contains the data you want to receive. Follow naming conventions.

In the TP READ instruction, specify the label of the buffer where the host data is to be stored. In the program, specify this buffer with a BUFFER instruction that contains the operand TPBSC.

You should also code error routines in TP READ to take over if an error or end condition occurs.

Submitting Background Jobs to the Host

Your program can allow the Series/1 to submit a host data set to the host batch job stream. To perform this function, code a TP SUBMIT instruction. The host data set can be either sequential or a member of a partitioned data set. In the TP instruction, specify the label of the TEXT instruction that contains the name of the host data set. In the program, code the TEXT instruction with the naming conventions for host data sets. The *Language Reference* contains the syntax and description of the TEXT instruction.

Performing Status Functions

Your program can perform the status functions associated with the system status data set that resides on the host system.

Writing Data to the System Status Data Set

You can set the occurrence of an event on the Series/1 by writing a record to the system status data set. To perform this function, code a TP SET instruction. In the TP SET instruction, refer to the label of the STATUS instruction, which references a record in the system status data set. In the program, code the STATUS instruction with its index entry and key fields, along with the optional 256-byte data field. The *Language Reference* contains the syntax and description of the STATUS instruction.

Retrieving a Record from the System Status Data Set

You can retrieve a specific record from the host data set and (optionally) read the record. To perform this function, code a TP FETCH instruction. In the TP FETCH instruction, refer to the label of the STATUS instruction that references the specific record of data in the system status data set. If you intend to read the record, code the "length" operand of TP FETCH with the number of bytes in the record to be read. If you do not want to read the record, you must still code the "length" operand, but enter the value zero.

Deleting a Record in the System Status Data Set

You can delete a record from the system status data set after you (optionally) read it. To perform this function, code a TP RELEASE instruction. This erases a Series/1 event that was set by a TP SET instruction. In the TP RELEASE instruction, refer to the label of the STATUS instruction which in turn refers to the specific record in the system status data set. If you intend to read the record before deleting it, you must code the "length" operand in TP RELEASE, specifying the record length. If you do not want to read it, you must still code the operand, specifying the value zero. In the program, code the STATUS instruction with the required index entry and key field.

Obtaining Time and Date from the Host

You can obtain the current time (hours, minutes, seconds) and date (day, month, year) from the host system. To perform this function, code a TP TIMEDATE instruction. You must specify the six-word data area where the time and date information is to be stored in the Series/1.

Sample Programs

The following sample programs show how to accomplish some of the HCF functions with the TP instructions.

Status Functions Sample Program

This program performs the SET, FETCH, and RELEASE function. It communicates with the system status data set on the host system.

```
PROGA    PROGRAM A                        PROGRAM A
STATA    STATUS  PROGID,KEYA              DEFINE STATUS ID & KEY
*
A        TP      SET,STATA                SEND MESSAGE TO PROGB
*                                          VIA HOST
A1       TP      FETCH,STATA,ERRORA       CHECK IF PROGB RECEIVED
*                                          MESSAGE
*     FALL THRU IF KEY & ID STILL ON HOST
*
         GOTO    A1                       CONTINUE INTERROGATION
ERRORA   EQU     *                        DELETE THE MESSAGE ON HOST
         PROGSTOP
         ENDPROG
         END
PROGB    PROGRAM B                        PROGRAM B
STATB    STATUS  PROGID,KEYA              DEFINE SAME STATUS ID & KEY
*
B        TP      FETCH,STATB,ERROR=ERRORB   FETCH MESSAGE
*
*     MESSAGE WAS FOUND AND IS DELETED, THUS SIGNALING PROGA
*
         TP      RELEASE,STATB
         GOTO    END
ERRORB   GOTO    B                        CONTINUE LOOKING FOR MESSAGE
END      PROGSTOP
         ENDPROG
         END
```

Figure 59. System Status Data Set Sample Program

Sample Program to Send Data to the Host

This same program sends a 256-byte Series/1 data set to a data set on the host. It prompts the user to specify the host data set to receive the data.

```
WRITASK  PROGRAM  TPOPEN,DS=((SOURCE,??))
*                 OPEN TP LINE
TPOPEN   READTEXT DSNAME,'HOST DATASET: ',PROMPT=COND
         TP       OPENOUT,DSNAME
         IF       (WRITASK,EQ,-1),GOTO,DSREAD          OPEN OK?
         MOVE     SWITCH,3                             ..TPOPEN ERROR
         GOTO     ERRSW
*                 READ A RECORD FROM DATA SET
DSREAD   READ   DS1,BUFFER,ERROR=ERR2,END=TPCLOSE
*                 WRITE A RECORD TO HOST
TPWRITE  TP     WRITE,BUFFER,256
         IF     (WRITASK,EQ,-1),GOTO,DSREAD           ..OK?
ERR1     MOVE   SWITCH,1                              ..WRITE ERROR
         GOTO   TPCLOSE
ERR2     MOVE   SWITCH,2                              ..READ ERROR
*     CLOSE DATA SET AND PRINT MESSAGE AS APPROPRIATE
TPCLOSE  TP     CLOSE
ERRSW    GOTO   (RET0,RET1,RET2,RET3),SWITCH
RET0     PRINTEXT '*****READ/WRITE SUCCESSFUL*****@'
         PROGSTOP
RET1     PRINTEXT '*****WRITE UNSUCCESSFUL*****@'
         PROGSTOP
RET2     PRINTEXT '*****READ UNSUCCESSFUL*****@'
         PROGSTOP
RET3     PRINTEXT '*****TP OPEN UNSUCCESSFUL*****@'
         PROGSTOP
SWITCH   DATA   F'0'
DSNAME   TEXT   LENGTH=40
BUFFER   BUFFER 256,TPBSC
         ENDPROG
         END
```

Figure 60. Sample Program to Send Data Set to the Host

Sample Program to Receive a Host Data Set

In this example, the Series/1 specifies that the host send it a data set. It reads the host data into a pre-allocated data set on a Series/1 volume. During program load, the user is prompted for the Series/1 data set where the host data will be placed.

```
READTASK PROGRAM  TPOPEN,DS=((TARGET,??))
*              OPEN TP LINE
TPOPEN   READTEXT DSNAME,'HOST DATASET: ',PROMPT=COND
         TP       OPENIN,DSNAME
         IF       (READTASK,EQ,-1),GOTO,TPREAD          OPEN OK?
         MOVE     SWITCH,3                              TP OPEN ERROR
         GOTO     ERRSW
*               READ A RECORD FROM HOST
TPREAD   TP   READ,BUFFER
         IF   (READTASK,EQ,-1),GOTO,DSWRITE             OK?
         IF   (READTASK,EQ,300),GOTO,TPCLOSE            END?
         GOTO ERR2
*                WRITE RECORD ON DISK
DSWRITE  WRITE DS1,BUFFER,ERROR=ERR1
         IF    (READTASK,EQ,-1),GOTO,TPREAD             OK?
ERR1     MOVE  SWITCH,1                                 WRITE ERROR
         GOTO ERRSW
ERR2     MOVE  SWITCH,2
*        CLOSE TP LINE AND PRINT MESSAGE AS APPROPRIATE
TPCLOSE  TP    CLOSE
ERRSW    GOTO  (RET0,RET1,RET2,RET3),SWITCH
RET0     PRINTEXT '*****READ/WRITE SUCCESSFUL*****@'
         PROGSTOP
RET1     PRINTEXT '*****WRITE UNSUCCESSFUL*****@'
         PROGSTOP
RET2     PRINTEXT '*****READ UNSUCCESSFUL*****@'
         PROGSTOP
RET3     PRINTEXT '*****TP OPEN UNSUCCESSFUL*****@'
         PROGSTOP
SWITCH   DATA  F'0'
DSNAME   TEXT  LENGTH=40
BUFFER   BUFFER 256,TPBSC
         ENDPROG
         END
```

Figure 61. Sample Program to Receive a Host Data Set

Interacting with the Host Communication Facility ($HCFUT1)

The $HCFUT1 utility allows the Host Communications Facility on the Series/1 to interact with the Host Communications Facility Installed User Program on the System/370. $HCFUT1 can perform four functions:

- Read a data set from the host
- Write a data set to the host
- Submit a job to the host
- Perform status functions in the system status data set.

Figure 62 lists the $HCFUT1 commands.

```
END       - END
FE        - FETCH STATUS
REL       - RELEASE STATUS
READDATA  - READ HOST
READ80    - READ 80 BYTE RECORDS - STORE 2/DISK RECORD
READOBJ   - READ 80 BYTE RECORDS - STORE 3/DISK RECORD
SE        - SET STATUS
SU        - SUBMIT A JOB
WR        - WRITE TO HOST
```

Figure 62. $HCFUT1 Commands

Notes:

1. See "Host Data Set Naming Conventions" on page CO-128 and"Host Data Set Characteristics" on page CO-129.
2. See "System Status Data Set" on page CO-130. Appendix B of the *Host Communications Facility Installed User Program* contains more details on its use.
3. The Host Communications Facility IUP, program number 5796-PGH, is required on the host System/370.
4. Host Communications Facility must be installed and configured on the Series/1.

Many of the functions that $HCFUT1 performs are the same as those you can program with the TP instructions.

Transferring Host Data to Series/1

You can transfer data with two commands, depending on the type of data being sent.

Interacting with the Host Communication Facility ($HCFUT1) *(continued)*

Using READDATA Command

The READDATA command transfers a data set from the host to the Series/1. The host logical record size is assumed to be 256 bytes.

The utility prompts for the following information:

- WORKFILE. This refers to the 1-8 character name of the Series/1 data set where the host data will be transferred, including the volume name.
- Record Count. This refers to the number of records to be transferred, beginning with the first. Use this if, for example, only the first 10 records of a 50-record data set are to be transferred.

  Enter zero to indicate that the entire data set is to be transferred.
- DSNAME. This refers to the name of the host data set to be transferred.

The following is a terminal printout of a typical run. In this example, all records (length = 256 bytes each) of the host data set "S1.EDX.TESTIN(DATA)" (which contains 40 records) are transferred to the Series/1 data set "DATAFIL2."

```
> $L $HCFUT1
LOADING $HCFUT1     8P,08.15.30, LP=4B00
WORKFILE(NAME,VOLUME):  DATAFIL2,EDX001

COMMAND (?):  READDATA
NO. OF RECORDS TO READ (0=ALL):  0
DSNAME:  S1.EDX.TESTIN(DATA)
END AFTER 40 RECORDS

COMMAND (?):
```

Using READ80 and READOBJ Commands

The READ80 and READOBJ commands transfer 80-byte records from a host data set and store them in 256-byte Series/1 disk or diskette data set records.

READ80 stores two 80-byte records per 256-byte disk record. The first 80-byte record is stored in the first 80 bytes of the disk record. The second 80-byte record is stored starting at byte 129 of the disk record. This format is compatible with the saved results of using $EDIT1N or $FSEDIT and is also the format required for input to a language compiler or $EDXASM program preparation. READ80 is normally used to transfer source program modules from the System/370 to Series/1 disk.

READOBJ stores three 80-byte records in the first 240 bytes of each disk record. This format is compatible with object modules produced by any of the assembler programs. It is also the format required for input to $EDXLINK and is one of the formats accepted by $UPDATE.

READOBJ is normally used to transfer the output object module of a host assembly to the Series/1 for processing by $EDXLINK or $UPDATE.

For both these commands, the utility prompts for the name of the Series/1 data set where the data is to go, the number of host records to be transferred, and the name of the host data set where the records come from.

Performing Status Functions

The status commands allow you to perform, from a terminal, the SET, FETCH, and RELEASE functions on the system status data set. The functions are identical to those you can perform with TP SET, TP FETCH and TP RELEASE.

For the SE and FE commands, the utility prompts for the index entry and key field . For the RE command, it prompts for the index entry. After performing one of the status functions, the utility sends a return code indicating the status of the data set.

Submitting Jobs to the Host Job Stream

The SU command allows you to submit a job to the host job stream. This function is identical to the one that TP SUBMIT performs. The utility prompts you for the name of the host data set you want to submit. Follow host data set naming conventions when you specify the name.

Sending Data to the Host

The WR command sends data from the Series/1 to the host. The host logical record size is assumed to be 256 bytes.

The utility prompts you for the following information:

- WORKFILE. This refers to the 1-8 character name of the Series/1 data set to be transferred, and its volume name, if not the IPL volume.
- Record Count. This refers to the number of records to be transferred, beginning with the first. This would be used if, for example, only the first 10 records of a 50-record data set are to be transferred.

  A count of zero is used to indicate that the entire data set is to be transferred.
- DSNAME. This refers to the name of the host data set to which the data is to be transferred. The name consists of up to 44 characters, or 54 characters if a member of a partitioned data set.

Interacting with the Host Communication Facility ($HCFUT1) *(continued)*

The following is a terminal printout of a typical run. In this example, 28 records of the Series/1 data set "DATAFIL1" are transferred to the host data set "S1.EDX.TESTOUT(DATA)."

```
 > $L $HCFUT1
WORKFILE(NAME,VOLUME):  DATAFIL1
$HCFUT1     8P,08.15.20, LP=4B00

COMMAND (?):  WR
NO. OF RECORDS TO WRITE(0=ALL):  28
DSNAME:  S1.EDX.TESTOUT(DATA)
END AFTER 28 RECORDS

COMMAND (?):
```

Part 2. Channel Attach

Part 2 discusses the Channel Attach Program, which allows Series/1 to communicate with large host systems.

Notes

Chapter 4. Channel Attach Program

The channel attach program allows the Series/1 to communicate with a larger host processor. This discussion refers to the System/370 as the host processor. However, the host can be any processor that uses the Basic Telecommunications Access Method (BTAM) form of data communications.

Series/1 is physically connected to the host by a channel attach device.

You must write the application programs that communicate with the System/370 host. The program must contain special Event Driven Language instructions, called CA instructions, used for the channel attach application. In addition, you can communicate with the host by using the $CHANUT1 utility.

Planning for the Channel Attach Application

Before you begin to write your application program to communicate with the host system, you should be familiar with the way channel attach works and what requirements and restrictions apply to its use.

Channel Attach Program ($CAPGM)

The channel attach program, as it executes on the Series/1, transfers data from the Series/1 application program to the channel attach device. At the same time, an application program on the System/370 is executing to transfer data between the channel attach device and the System/370.

$CAPGM allows you to perform the following functions through your application program running on the Series/1:

- Establishing, controlling, and terminating access between Series/1 and System/370
- Transferring data between Series/1 and System/370
- Communicating with System/370 application programs by 32 data ports (System/370 device addresses)
- Handling interrupts from the channel attach device
- Managing data ports to avoid conflict or contention
- Performing error recovery and retry
- Performing error logging
- Tracing Series/1 I/O commands and attention interrupts from the channel attach device.

Channel Attach Device (4993)

The channel attach device provides an interface between a Series/1 and a System/370. It responds to commands from the System/370 and directs the information to the Series/1. Likewise, the device responds to commands from the Series/1 and directs the information to the host.

The connection between the channel attach device and the 370 consists of 32 device address (data port) connections. The channel attach device allows activity over only one port at a time.

Software Considerations

During system generation for the Series/1, you must define each channel attach device as an EXIO device. The instructions for this procedure are described in the *Installation and System Generation Guide.*

Hardware Considerations

You should consider the following when installing your channel attach device:

- By installing the channel attach device in the I/O expansion unit rather than the processor, you will be able to turn off the processor power temporarily without having to turn off the channel attach device first. If the channel attach device is in the processor, you *must* follow the power-off sequence described in "Powering On The Channel Attach Device" on page CO-148 to power off.

- When you install your channel attach device, it must be jumpered correctly to specify the lowest System/370 device address you will be using. For example, if your channel attach device is connected to the host on channel 5, subchannel 8 (with 32 device addresses X'580' to X'59F'), then the setting of the jumpers on the channel attach feature card should be for X'80'. You must be able to provide this information to the person who installs your channel attach device.

- On the System/370, your channel attach device must be defined to use a shared unit control word (UCW). Failure to specify a shared UCW can cause unpredictable results when you are using more than one port.

Tailoring the Channel Attach Program

During system generation for the Series/1, you can tailor the channel attach program to fit your specific needs by editing the program's control module, $CACBS. The section of $CACBS that you should edit consists of channel attach control block statements, CACB1 and CACB2. Two channel attach control block statements are required for each channel attach feature installed (physically and logically) on the Series/1 processor. CACB1 and CACB2 are configuration statements that create control blocks. These statements describe to the channel attach program the characteristics of a particular channel attach device. The CACB1 and CACB2 statements for a device must be contiguous and both must specify the same device address, trace size, and number of ports. In these statements, you can specify the following information:

- device address of the channel attach device
- amount of storage for the channel attach trace area
- number of host ports with which the device will connect
- number of retries during errors.

After you modify the CACB1 and CACB2 statements, assemble the $CACBS module and use $EDXLINK or $LINK/$UPDATE to link edit it to the channel attach support modules. If you followed the standard installation procedures, the link-edit control statements to do this are in data set $CALNK on EDX002.

$CALNK provides full support for a channel attach system, including trace, log, and OLTEP support. You should use the full support system for initial debugging efforts.

To modify the provided support, you may want to include XCAXDIAG instead of $CAXDIAG to eliminate OLTEP support; XCAXLOG instead of $CAXLOG to eliminate error logging; XCAXTRCE instead of $CAXTRCE to eliminate tracing; and XCAXERR instead of $CAXERR to eliminate the task error exit facility.

Channel Attach Program

Planning for the Channel Attach Application *(continued)*

Powering On The Channel Attach Device

Before any communication can occur, you must power on the channel attach device at the Series/1. Use the following procedure:

1. Turn off the channel attach device, and ensure that it is offline. Set the Enable/Disable switch on the 4993 Termination Enclosure to the Disable position and the On/Off switch to the Off position.

2. Turn on the other devices. Set the On/Off switch for each Series/1 unit (except for the 4993 Termination Enclosure and the Series/1 processor) to the On position.

3. Turn on the processor by setting the On/Off switch on the Series/1 processor to the On position.

4. Turn on the Termination Enclosure unit by setting the On/Off switch on the 4993 Termination Enclosure to the On position and making sure that the Power On indicator is on.

5. After you complete the above steps, place the channel attach device online to the System/370 by setting the Enable/Disable switch on the 4993 Termination Enclosure to the Enable position. Make sure that the Disable indicator is off (indicating that the unit is online), and notify the System/370 operator that the channel attach device is online. If you attempt to vary a System/370 device address online when the channel attach device is not powered on and enabled, the System/370 receives a NO PHYSICAL PATH message.

To power off the channel attach device, use the reverse sequence of the power on instructions. You must disable the channel attach device, turn off the device, and turn off the I/O expansion unit and the other devices. Never turn the channel attach device off when the device is enabled.

Programming for the Channel Attach Application

This section tells how to write programs that communicate between the Series/1 and the host by means of the channel attach.

Event Driven Executive Instruction Set

Your program must contain these channel attach instructions. They provide you with the interface to the System/370 channel attach program. They are:

- CACLOSE - Close a channel attach port
- CAIOCB - Create a channel attach port I/O control block
- CAOPEN - Open a channel attach port
- CAPRINT - Print channel attach trace data
- CAREAD - Read from a channel attach port

- CASTART - Start channel attach device
- CASTOP - Stop a channel attach device
- CATRACE - Control channel attach trace
- CAWRITE - Write to a channel attach port.

For the syntax of each instruction, refer to the *Language Reference.*

Detecting and Handling Errors

Each instruction sets the task code word to indicate success or failure of the operation it performs. You should use the ERROR operand of the instruction or check the task code word after each instruction.

To ensure that the instructions are successful, your program should wait on an event control block (ECB) for completion of the I/O associated with the instruction. CATRACE has no I/O associated with it so no wait is required. CASTOP, CASTART, and CAPRINT use the ECB supplied with the instruction. CAOPEN, CAREAD, CAWRITE, and CACLOSE use the first three words of the CAIOCB as the ECB to wait on. After you do a WAIT on the ECB, check the completion code for I/O errors. Do not issue a WAIT if the return code in the task code word indicates an unsuccessful operation. The second word of the TCB contains the address of the instruction that received the error.

BTAM Considerations

The Basic Telecommunications Access Method (BTAM) interfaces with the System/370 channel attach; you should consider the following:

- BTAM issues an ERASE/WRITE of 1 byte (X'C3') or (X'7B') when you open or close a channel attach port. Respond with a read of this ERASE/WRITE when opening a port. When closing a port, you can either read the (X'C3') or ignore it. If you ignore the (X'C3') then you may get a return code from the close operation to show data pending from the host.

- If BTAM requests an I/O operation to the device and the device is in a "not ready" condition (the System/370 device address has not been enabled by Series/1), BTAM posts an intervention required (X'41').

- On an asynchronous device end from the Series/1 (caused by a Series/1 ENABLE System/370 DEVICE ADDRESS), the System/370 application may elect to be notified by BTAM (OS/VS only). This is done by specifying the READYQ option on the DCB macro, which causes the user to be posted when a device end occurs for the specified device address. The device end is ignored if the READYQ option is not selected or if using DOS/VS BTAM.

- BTAM issues retries for read errors and all busy conditions.

- You must not issue any BTAM macro which might cause BTAM to generate a channel program that contains both read and write channel command words (CCWs), such as Read Modify Position or Read Buffer Position. Extraneous I/O operations that result from chained read and write CCWs can invalidate protocol understanding at the Series/1.
- Do not issue a command that requires an attention interrupt to begin on the System/370 as the first command after OPEN. If you do, you must have some method of ensuring that the System/370 opening process is done before you issue the (Series/1) attention interrupt.
- The first three bytes of data sent by a Series/1 write request can have explicit meaning to the System/370 support program. Therefore, be careful when setting the first three bytes of data, because unpredictable results can occur. To avoid problems, set the first three bytes of the data to X'7D4040'. This corresponds to an attention identification descriptor (AID) and two bytes of null buffer address. The X'7D' corresponds to an Enter key response from an IBM 3272 Control Unit.

Assembling the Application Program

If you are going to assemble your program with either the host or native macro assemblers, you don't have any special steps to perform. If you are using $EDXASM to assemble, you must code the following statements in your program for proper assembly.

```
COPY CMDEQU
COPY PROGEQU
```

These statements generate several pages of equates. To suppress printing these equates, code the following statements:

```
PRINT OFF
COPY  CMDEQU
COPY  PROGEQU
PRINT ON
```

Link-Editing the Application Program

After assembling your program, you must use $EDXLINK or $LINK/$UPDATE to link-edit it. If your channel attach support has been installed as shown in the *Program Directory*, the link editing control statements required to link your application program are in the data set USERPGM on EDX002. A listing of the data set USERPGM follows.

```
 LIST OF DATA SET USERPGM ON EDX002
*
* EVENT DRIVEN EXECUTIVE _ SYSTEM/370 CHANNEL ATTACH VER. 1
*****************************************************************
* COMMENTS MAY BE INCLUDED BY AN '*' IN COLUMN 1                *
* USE THIS TECHNIQUE TO OMIT UNNEEDED SUPPORT                   *
*****************************************************************
 OUTPUT   <---- YOUR OUTPUT MODULE NAME AND VOLUME IT RESIDES
                 ON, GOES HERE
*
 INCLUDE  <---- YOUR INPUT MODULE NAME AND VOLUME IT RESIDES
                 ON, GOES HERE
 INCLUDE  $CAPRCES,EDX002      *MAKE A COMMENT AND INCLUDE
*INCLUDE  XCAPRCES,EDX002      *THIS IF TRACE PRINT OMITTED
 INCLUDE  $CABEGIN,EDX002
 INCLUDE  $CAPARM,EDX002
 INCLUDE  $CALOGC0,EDX002      *MAKE A COMMENT AND INCLUDE
*INCLUDE  XCALOGC0,EDX002      *THIS IF INPUT ERR. OK OMITTED
 INCLUDE  $CALOGIC,EDX002
 INCLUDE  $CAFILL,EDX002
 INCLUDE  $CAPRNT,EDX002
                               *MAKE THIS A COMMENT IF TRACE
                               *PRINTING OMITTED
 INCLUDE  $$SVC,ASMLIB
 INCLUDE  $$RETURN,ASMLIB
 INCLUDE  $EDXATSR,ASMLIB
 END

LIST COMPLETE
```

Figure 63. Listing of USERPGM Data Set

The name of your output module and input module refer to your object output and input data sets respectively.

This link editing control data set provides you with the full support link modules. You should use the full support system during initial debugging activities.

Starting a Channel Attach Device

Code a CASTART instruction to load the channel attach device support program and prepare the channel attach device to accept interrupts from the host. You must start the channel attach device before you can open any of its ports and before you can issue any I/O instructions.

After your program issues a CASTART instruction, check the return code in the first word of the task control block (TCB). If the return code indicates a successful request, issue a WAIT for the I/O to complete. Use the event control block (ECB) operand of the CASTART instruction to wait on I/O completion. If the return code indicates that the device was already started or an error occurred, do not issue a WAIT instruction.

Programming for the Channel Attach Application *(continued)*

Opening a Channel Attach Port

Code a CAOPEN instruction to open a specific port on the channel attach device. This logically assigns the port to your application program, and enables it to accept interrupts from the host. You must open a port before using it for data transfer.

You must code a control block instruction (CAIOCB) for each port you open in your program. The CAIOCB stores the device address, port number, control block addresses associated with the port, and the ECB used to wait for I/O completion on the port. Its use it discussed in the next section.

Coding the Control Block for a Channel Attach Port

The CAIOCB statement creates a channel attach port I/O control block which contains the information required to access a port. It is a nonexecutable instruction which allocates storage. You supply the device address, port number, and the address of the first buffer control area. Other information in the System/370 channel attach I/O control block is supplied by the System/370 channel attach link module when the device is opened. In every other CA instruction you code to perform operations on a port, you must refer to the label of the CAIOCB for that port, which you code in the nonexecutable section of your program.

As for all EDX channel attach instructions, check the return code in the TCB before issuing a WAIT.

Issuing I/O

Your application program requests I/O processing by issuing CAREAD and CAWRITE instructions.

For both CAREAD and CAWRITE, check the return code in the TCB before issuing a WAIT instruction for the I/O to complete.

Series/1 Receiving Data from the Host (CAREAD)

The CAREAD instruction reads data from a host device address. Specify the CAIOCB statement that refers to the port in the CAREAD instruction. Each CAREAD instruction must supply the addresses of two buffer control blocks.

When you issue a Series/1 read request, one of two conditions are true: either the System/370 has already issued the write to match your CAREAD, or your CAREAD was issued before the matching write request was received. If the System/370 has already issued its write request, your CAREAD is posted immediately and the address of the buffer that the System/370 wrote to is returned to you. If the System/370 has not issued a write to match your CAREAD, the channel attach program holds your CAREAD until the System/370 issues its write request.

Because the System/370 can issue a write request at any time, a buffer must be available to receive data from the System/370. This is accomplished as follows:

- When you open a channel attach port, you must point to a buffer control block defining the address, size, and partition of the buffer to receive data for the first System/370 write. This information is stored by the channel attach program and is returned to you when your first CAREAD is complete.

- When you issue a CAREAD instruction it must identify two buffer control blocks:

  – The first buffer control block receives (from the channel attach program the address, size, and partition of the data buffer written by the System/370 to satisfy that CAREAD.

  – The second buffer control block must contain three words defining the address, size and partition of the buffer to be used for the next System/370 write. These values are stored by the channel attach program, and are returned to you when the next Series/1 CAREAD is complete.

On every Series/1 CAREAD instruction, you are told where the data for the read was stored, and you supply the information required to set up for the next System/370 write operation.

Series/1 Sending Data to the Host

The CAWRITE instruction sends data from your application program buffer to a channel attach device port. On the CAWRITE instruction, you must specify the CAIOCB used to open the port.

When a Series/1 CAWRITE is issued to a port and the System/370 application program is not actively reading from the corresponding device address, the channel attach program issues an attention interrupt to the System/370. This attention interrupt notifies the System/370 application program that the data from the Series/1 is available for the associated device address.

The System/370 should then select the port and issue a READ. The channel attach program recognizes the READ command and issues a start I/O (write) command to the channel attach device to start transfer of the contents of the Series/1 data buffer to the System/370.

When data transfer is complete, the write operation is not posted until the System/370 application program acknowledges the data transfer or indicates negative acknowledgement.

This acknowledgement can be in one of two forms. If the System/370 program issues an Erase All Unprotected (EAU) command after data has been sent to the host, then the EAU is considered to be a positive acknowledgement of the data transfer. The CAWRITE which caused the data to be sent to the host is then posted.

A second way the System/370 acknowledges a CAWRITE is by issuing a write command. When a System/370 Write or Erase/Write is received as an acknowledgement of a CAWRITE, bit 6 of the first data byte from the System/370 is examined to determine whether it is a positive acknowledgement (bit 6 on) or a negative acknowledgement (bit 6 off), and the Series/1 program is posted appropriately.

If the System/370 issues additional read requests before it acknowledges the Series/1 CAWRITE, the read requests connect to the same CAWRITE, causing retransmission of the data.

If the System/370 tries to read when no corresponding Series/1 CAWRITE has been issued, the channel attach program generates a write operation and sends X'604040' to the host; this indicates that a CAWRITE is not pending on the Series/1. If the channel attach program has sent X'604040' to the host one or more times before the Series/1 user issues a CAWRITE, then when the CAWRITE is issued the channel attach program sets the appropriate flags to allow the next System/370 read to the Series/1 user data buffer. No attention interrupt is issued to the System/370 in this case.

The first three bytes of data sent by a Series/1 CAWRITE can have explicit meaning to the System/370 support program. Therefore, be careful when setting the first three bytes of data, because unpredictable results can occur. To avoid problems, set the first three bytes of the data to X'7D4040'. This corresponds to an attention identification descriptor (AID) and two bytes of null buffer address. The X'7D' corresponds to an Enter key response from an IBM 3272 Control Unit.

Closing a Channel Attach Port (CACLOSE)

When your program no longer requires a port it should issue a CACLOSE instruction to free the port. A CACLOSE for a port causes the channel attach program to:

- Reinitialize the control blocks for the port so the port can be opened by another application program.
- Disable the port except for port 0 which is kept open for potential use by the Online Test Executive Program (OLTEP).

Stopping the Channel Attach Device (CASTOP)

The CASTOP instruction frees the channel attach device. A CASTOP causes the channel attach program to:

- Disable port 0
- Disable the channel attach device for interrupts
- Reinitialize the control block for the channel attach device so it can be started by another application program
- Unload the channel attach program (only if all channel attach devices are stopped).

Tracing Series/1 I/O during Channel Attach (CATRACE)

The CATRACE instruction enable or disables the collection of I/O trace data for a channel attach device. Channel attach trace data is collected in processor storage and, for performance reasons, should only be used during debugging.

Printing Channel Attach Trace Data (CAPRINT)

To print the entire area of trace data obtained through CATRACE, code a CAPRINT instruction. The data prints out on a printer or displays at your terminal. Tracing is disabled while printing is in progress.

Interacting with Channel Attach (Using $CHANUT1 Utility)

The channel attach utility, $CHANUT1, allows you to perform several functions associated with channel attach operations:

- start or stop channel attach device
- enable or disable I/O tracing
- print trace data.

Invoke $CHANUT1 by the $L command or by the session manager. You can load the utility into any partition. As soon as it is loaded, the utility asks for the address of the channel attach device you want to work with. All commands you enter during a session with $CHANUT1 will apply to the first device you specify, unless you change the address (this procedure is discussed below). The $CHANUT1 commands interface with the channel attach program in the same manner as the channel attach instructions. The error codes for the $CHANUT1 commands are the same as those for the corresponding instructions.

$CHANUT1 Commands

The $CHANUT1 commands are shown below. To obtain this list at your terminal, enter a question mark in response to the prompting message, COMMAND(?):

```
COMMAND(?):  ?

    CA  --    CHANGE DEVICE ADDRESS
    EN  --    TERMINATE THE UTILITY
    PR  --    PRINT THE TRACE AREA
    ST  --    START A CHANNEL ATTACH DEVICE
    SP  --    STOP A CHANNEL ATTACH DEVICE
    TR  --    ENABLE/DISABLE TRACING
COMMAND(?):
```

Interacting with Channel Attach (Using $CHANUT1 Utility) *(continued)*

Changing Channel Attach Device Address (CA)

The CA command changes the address of the channel attach device you want to use during the $CHANUT1 session.

Ending the Utility (EN)

The EN command ends the $CHANUT1 utility.

```
COMMAND(?):  EN
```

Printing the Trace Data (PR)

The PR command prints the trace buffer, with the title you enter, on a terminal.

```
COMMAND(?):  PR
TITLE:  TRACE PRINTOUT 07/26/80
CONSOLE:  $SYSPRTR
PRINT TRACE BUFFER SUCCESSFUL
COMMAND(?):
```

Stopping a Channel Attach Device (SP)

The SP command stops the channel attach device you have specified.

```
COMMAND(?):  SP
STOP DEVICE SUCCESSFUL
COMMAND(?):
```

Starting a Channel Attach Device (ST)

The ST command starts the channel attach device you have selected.

```
COMMAND(?):  ST
START DEVICE SUCCESSFUL
COMMAND(?):
```

Performing Trace Function (TR)

The TR command allows you to enable (E) or disable (D) the trace function.

```
COMMAND(?):  TR
ENABLE OR DISABLE:  D
ENABLE/DISABLE SUCCESSFUL
COMMAND(?):
```

Channel Attach Sample Programs

This section contains two sample programs. The first, executes on a Series/1 using the EDX channel attach support. It communicates with an OS/VS2 application program which is executing at the same time on the host System/370.

The second sample program executes on a System/370 using BTAM or BTAM-ES facilities. It communicates with the first sample program, which is running on the Series/1.

Configuration Requirements for Sample Programs

- Hardware
  - System/370

    Block multiplexer or selector channel

    One control unit position on channel

    Other peripherals to support OS/VS2.
  - Series/1

    IBM Series/1 hardware required to operate EDX

    IBM 4993-1 Series/1-System/370 termination enclosure

    IBM Series/1-System/370 Channel Attachment Feature #1200.
- Software
  - System/370 software:

    OS/VS2 (MVS)

    Basic Telecommunications Access Method (BTAM)

    Channel attach device defined to OS/VS2 via I/O generation

    User application program.
  - Series/1 software:

    EDX operating system (any version)

    EDX Channel Attach Program

    User application program.

Channel Attach Program

Channel Attach Sample Programs *(continued)*

General Guide for Execution of Sample Programs

- Install channel attach support.
- Modify the sample program if your channel attach device is not at address 10. Assemble, link, and update Series/1 program.
- Assemble host program and link-edit for S/370 execution.
- Power on the channel attach device and set the enable/disable switch to enable.
- Start sample program on the Series/1.
- When prompted, start the sample program on the System/370.

Series/1 Sample Program

The Series/1 sample program (SAMPLEA) performs the following functions:

- Starts the channel attach device (the CASTART instruction)
- Enables and disables I/O tracing (the CATRACE instruction)
- Opens channel attach device port #1 (the CAOPEN instruction)
- Reads from the System/370 over port #1 (the CAREAD instruction)
- Writes to the System/370 over port #1 (the CAWRITE instruction)
- Closes channel attach device port #1 (the CLOSE instruction)
- Prints the I/O trace area (the CAPRINT instruction)
- Stops the channel attach device (the CASTOP instruction).

```
SAMPLEA  PROGRAM BEGIN
         PRINT OFF
         COPY PROGEQU            ===> REMOVE FOR MACRO ASSEMBLER
         COPY CMDEQU             ===> REMOVE FOR MACRO ASSEMBLER
         PRINT ON
BEGIN    PRINTEXT '@THIS IS A TEST OF THE CHANNEL ATTACH SUPPORT.'
         PRINTEXT '@THE CHANNEL ATTACH DEVICE MUST BE ON AND'
         PRINTEXT '@ENABLED BEFORE YOU PRESS ENTER TO CONTINUE.'
         READTEXT SYNC
*********************************************************************
*  START CHANNEL ATTACH DEVICE 10                                   *
*********************************************************************
         CASTART 10,STREVNT,ERROR=STRTERR
         WAIT  STREVNT
         IF    (STREVNT,NE,+MINUS1)
           GOTO  STRTERR
         ENDIF
         PRINTEXT '@DEVICE 10 STARTED'
*********************************************************************
*  TURN ON I/O TRACING                                              *
*********************************************************************
         CATRACE 10,ENABLE=YES,ERROR=TRCERR
         PRINTEXT '@TRACE ENABLED'
*********************************************************************
*  OPEN PORT ONE                                                    *
*********************************************************************
         CAOPEN PORTONE,ERROR=OPNERR
         WAIT  PORTONE
         IF    (PORTONE,NE,+MINUS1)
           GOTO  OPNERR
         ENDIF
         PRINTEXT '@PORT ONE OPENED'
         PRINTEXT '@PLEASE START SAMPLEC ON THE SYSTEM/370.'
         PRINTEXT '@PRESS ENTER WHEN RDACK APPEARS ON THE '
         PRINTEXT '@S/370 DISPLAY, INDICATING THAT THE S/370'
         PRINTEXT '@HAS COMPLETED OPEN PROCESSING.'
         READTEXT SYNC
```

Figure 64 (Part 1 of 6). Series/1 Sample Program

```
*********************************************************************
*  READ C3 WRITTEN BY SYSTEM/370 DURING OPEN PROCESSING             *
*********************************************************************
         CAREAD PORTONE,WHATREAD,INAREA,ERROR=C3ERR
         WAIT  PORTONE
         IF    (PORTONE,NE,+MINUS1)
           GOTO  C3ERR
         ENDIF
         PRINTEXT '@READ OF C3 SUCCESSFUL'
*********************************************************************
*  WRITE MESSAGE TO SYSTEM/370 TO ACKNOWLEDGE RECEIPT OF C3         *
*********************************************************************
         CAWRITE PORTONE,OUTACK,ERROR=ACKERR
         WAIT  PORTONE
         IF    (PORTONE,NE,+MINUS1)
           GOTO  ACKERR
         ENDIF
         PRINTEXT '@ACK OF C3 WRITTEN TO SYSTEM/370'
*********************************************************************
*  I/O LOOP                                                         *
*********************************************************************
         DO    WHILE,(COUNT,LT,+LIMIT)
           ADD   COUNT,1
*********************************************************************
*  WRITE MESSAGE TO SYSTEM/370                                      *
*********************************************************************
           CAWRITE PORTONE,OUTAREA,ERROR=WRITERR
           WAIT  PORTONE
           IF    (PORTONE,NE,+MINUS1)
             GOTO  WRITERR
           ENDIF
           PRINTEXT '@DATA WRITTEN TO SYSTEM/370'
```

Figure 64 (Part 2 of 6). Series/1 Sample Program

```
*********************************************************************
*  READ MESSAGE FROM SYSTEM/370                                     *
*********************************************************************
           CAREAD PORTONE,WHATREAD,INAREA,ERROR=READERR
           WAIT  PORTONE
           IF    (PORTONE,NE,+MINUS1)
             GOTO  READERR
           ENDIF
           PRINTEXT '@DATA READ FROM SYSTEM/370'
*********************************************************************
*  WRITE ACK TO SYSTEM/370                                          *
*********************************************************************
           CAWRITE PORTONE,OUTACK,ERROR=RDACKERR
           WAIT PORTONE
           IF   (PORTONE,NE,+MINUS1)
             GOTO  RDACKERR
           ENDIF
           PRINTEXT '@ACK WRITTEN TO SYSTEM/370'
         ENDDO
*********************************************************************
*  CLOSE PORT ONE                                                   *
*********************************************************************
EXIT       CACLOSE PORTONE,ERROR=CLOSERR
           WAIT  PORTONE
           IF    (PORTONE,NE,+MINUS1)
             GOTO  CLOSERR
           ENDIF
           PRINTEXT '@PORT ONE CLOSED'
```

Figure 64 (Part 3 of 6). Series/1 Sample Program

```
*********************************************************************
*  TURN OFF TRACE                                                   *
*********************************************************************
TRCDIS   CATRACE 10,ENABLE=NO,ERROR=TDISERR
         PRINTEXT '@TRACE DISABLED'
*********************************************************************
*  PRINT THE TRACE AREA ON THE SYSTEM PRINTER                       *
*********************************************************************
PRNTRCE  CAPRINT 10,STREVNT,TITLE=TITLDATA,ERROR=PRNTERR
         WAIT  STREVNT
         IF    (STREVNT,NE,+MINUS1)
           GOTO  PRNTERR
         ENDIF
         PRINTEXT '@TRACE AREA PRINTED'
*********************************************************************
*  STOP THE CHANNEL ATTACH DEVICE                                   *
*********************************************************************
STOPDEV  CASTOP 10,STREVNT,ERROR=STOPERR
         WAIT  STREVNT
         IF    (STREVNT,NE,+MINUS1),AND,(STREVNT,NE,+ENDED)
           GOTO  STOPERR
         ENDIF
         PRINTEXT '@DEVICE 10 STOPPED'
         PROGSTOP -1
```

Figure 64 (Part 4 of 6). Series/1 Sample Program

```
*********************************************************************
*  DATA AREAS AND I/O BUFFERS                                       *
*********************************************************************
PORTONE  CAIOCB 10,PORT=1,BUFFER=C3AREA   CAIOCB FOR PORT ONE
STREVNT  ECB    0                 EVENT FOR START, STOP, PRINT
C3AREA   DATA   A(INBUFF)         BUFFER CONTROL AREA FOR
         DATA   F'1'              * READ OF C3
         DATA   F'0'              * FROM S/370 OPEN
INAREA   DATA   A(INBUFF)         BUFFER CONTROL AREA FOR
         DATA   F'128'            * DATA INPUT
         DATA   F'0'              * FROM S/370 WRITE
OUTAREA  DATA   A(OUTBUFF)        BUFFER CONTROL AREA FOR
         DATA   F'128'            * WRITE OF DATA
         DATA   F'0'              * TO SYSTEM/370
OUTACK   DATA   A(OUTBUFF)        BUFFER CONTROL AREA FOR
         DATA   F'4'              * WRITE OF ACK
         DATA   F'0'              * TO SYSTEM/370
WHATREAD DATA   3F'0'             RECEIVE CONTROL INFO FOR READ
INBUFF   DATA   128C'0'
OUTBUFF  DATA   X'7D4040AA'
         DATA   C'BB CC DD EE FF GG HH'
         DATA   C'11 22 33 44 55 66 77'
         DATA   C'ZZZZZYYYYYXXXXXWWWWW'
         DATA   C'THIS IS TO BE WRITTE'
         DATA   C'N TO THE SYSTEM/370.'
         DATA   C' THE END'
COUNT    DATA   F'0'
LIMIT    EQU    2
MINUS1   EQU    -1
SYNC     TEXT   LENGTH=4
TITLDATA DATA   A(MYTITLE)
         DATA   F'25'
MYTITLE  DATA   C'SAMPLE PROGRAM TRACE AREA'
ENDED    EQU    599
```

Figure 64 (Part 5 of 6). Series/1 Sample Program

```
*********************************************************************
*  ERROR MESSAGES                                                   *
*********************************************************************
STRTERR  PRINTEXT '@ERROR DURING START PROCESSING'
         GOTO EXIT
OPNERR   PRINTEXT '@ERROR DURING OPEN PROCESSING'
         GOTO EXIT
C3ERR    PRINTEXT '@ERROR DURING C3 PROCESSING'
         GOTO EXIT
ACKERR   PRINTEXT '@ERROR DURING C3/ACK PROCESSING'
         GOTO EXIT
WRITERR  PRINTEXT '@ERROR DURING S/1 WRITE PROCESSING'
         GOTO EXIT
READERR  PRINTEXT '@ERROR DURING S/1 READ PROCESSING'
         GOTO EXIT
RDACKERR PRINTEXT 'ERROR DURING READ/ACK PROCESSING'
         GOTO EXIT
TRCERR   PRINTEXT '@ERROR DURING TRACE ENABLE'
         GOTO EXIT
TDISERR  PRINTEXT '@ERROR DURING TRACE DISABLE'
         GOTO PRNTRCE
PRNTERR  PRINTEXT '@ERROR DURING PRINT OF TRACE AREA'
         GOTO STOPDEV
CLOSERR  PRINTEXT '@ERROR DURING CLOSE PROCESSING'
         GOTO TRCDIS
STOPERR  PRINTEXT '@ERROR DURING STOP PROCESSING'
         PROGSTOP -1
         ENDPROG
         END
```

Figure 64 (Part 6 of 6). Series/1 Sample Program

Host Sample Program

This program executes on a System/370 using OS/VS Basic Telecommunications Access Method (BTAM) facilities, to communicate the application program executing at the same time on the Series/1. Refer to the previous program, which is the companion to this host program.

```
*********************************************************************
*         INVOKE SAMPLEC VIA TSO COMMANDS OR JCL.                   *
*   EXAMPLE: INVOKE SAMPLEC VIA TSO COMMANDS:                       *
*   FREE FI(PRINTER,SNAP,S1GROUP)                                   *
*   ALLOC FI(SYSABEND) DA(DUMP.LIST) NEW SPACE(5,1) CYLINDERS +     *
*         CATALOG                                                   *
*   ALLOC FI(SNAP) DA(SNAP.LIST) NEW SPACE(5,1) CYLINDERS CATALOG   *
*   ALLOC FI(S1GROUP) UNIT(581) NEW                                 *
*   ALLOC FI(PRINTER) DA(*)                                         *
*   CALL 'PROJECT.LIB.LOAD(SAMPLEC)'                                *
*   FREE FI(PRINTER,SNAP,S1GROUP)                                   *
*   EXAMPLE: INVOKE SAMPLEC VIA OS/VS JCL:                          *
*  //        JOB (ACCOUNTING INFO),'NAME', ...                      *
*  //SAMPLEC EXEC  PGM=SAMPLEC                                      *
*  //STEPLIB DD  DISP=SHR,DSN=PROJECT.LIB.LOAD                      *
*  //S1GROUP DD  UNIT=581,DISP=NEW                                  *
*  //PRINTER DD  SYSOUT=A                                           *
*  //SNAP    DD  SYSOUT=A                                           *
*  //SYSIN   DD  *                                                  *
*  /*                                                              *
*********************************************************************
*     EQUATES                                                       *
*********************************************************************
R0        EQU   0                        TEMPORARY STORAGE
R1        EQU   1                        TEMPORARY STORAGE
RLN       EQU   1                        RELATIVE LINE NUMBER
R2        EQU   2                        MESSAGE ADDRESS
R3        EQU   3                        LOOP COUNTER
R4        EQU   4                        BAL
R5        EQU   5                        BAL
DCBREG    EQU   6                        DCB USING REGISTER
DECBREG   EQU   7                        DECB USING REGISTER
R8        EQU   8                        BAL
R9        EQU   9                        NOT USED
R10       EQU   10                       NOT USED
BASEREG   EQU   11                       BASE USING REGISTER
R12       EQU   12                       NOT USED
SAVEREG   EQU   13                       SAVE AREA ADDRESS REGISTER
R14       EQU   14                       LINKAGE
R15       EQU   15                       LINKAGE
```

Figure 65 (Part 1 of 8). Host Sample Program

```
*********************************************************************
* START SAMPLEC                                                     *
*********************************************************************
SAMPLEC  CSECT                          SAMPLEC PROGRAM
         SAVE  (14,12)                  SAVE REGISTERS
         BALR  BASEREG,0                ESTABLISH ADDRESSABILITY
         USING *,BASEREG                     FOR CSECT.
         USING IHADCB,DCBREG            ESTABLISH ADDRESSABILITY
         USING IECTDECB,DECBREG              FOR DCBS AND DECBS
         ST    SAVEREG,SAVEAREA+4       STORE ADDRESS OF SAVE AREA
         LA    SAVEREG,SAVEAREA         LOAD THIS PROG SAVE AREA ADDR
         LA    DECBREG,CADECB           LOAD DECB USING REG
*********************************************************************
*  OPEN PRINTER DATASET                                             *
*********************************************************************
         LA    DCBREG,PRINTDCB          LOAD DCB DSECT REG
         OPEN  (PRINTDCB,(OUTPUT))      OPEN PRINTER DCB
         TM    DCBOFLGS,X'10'           IF OPEN OK
         BO    CAOPEN1                    BRANCH
         ABEND 1,DUMP                   ELSE ABEND
*********************************************************************
*          ISSUE OPEN MACRO FOR SNAP DATASET * * * *
*********************************************************************
CAOPEN1  LA    DCBREG,SNAPDCB           LOAD DCB DSECT REG
         OPEN  (SNAPDCB,(OUTPUT))       OPEN SNAP DCB
         TM    DCBOFLGS,X'10'           IF OPEN OK
         BO    CAOPEN2                    BRANCH
         ABEND 2,DUMP                   ELSE ABEND
*********************************************************************
*  OPEN CHANNEL ATTACH DEVICE DCB                                   *
*********************************************************************
CAOPEN2  LA    DCBREG,CADCB             LOAD DCB DSECT REG
         MVC   COP(8),OPEN              CURRENT OP = OPEN
         OPEN  (CADCB)                  OPEN SERIES/1 LINE GROUP DCB
         TM    DCBOFLGS,X'10'           IF OPEN OK
         BO    CAINIT                     BRANCH
         LA    R2,MSG09                 ELSE
         BAL   R5,PRINTIO                   PRINT MESSAGE
         ABEND 3,DUMP                   ELSE ABEND
CAINIT   EQU   *
         LA    R2,MSG00                 PRINT BEGIN
         BAL   R5,PRINTIO               TEST MESSAGE.
         LA    R2,MSG01                 PRINT CHANNEL ATTACH
         BAL   R5,PRINTIO               DEVICE OPEN MESSAGE.
```

Figure 65 (Part 2 of 8). Host Sample Program

```
*********************************************************************
* READ ACKNOWLEDGE MESSAGE FROM SERIES/1 FOR BTAM OPEN'S WRITE.
*********************************************************************
         BAL   R8,RDACK                READ S/1 ACK MSG
         BAL   R4,CHKRTN               CHECK RETURN CODE
*********************************************************************
*         DO UNTIL COUNT = LIMIT
*********************************************************************
         LA    R3,NLIMIT               SET R3 = LIMIT
LOOP     EQU   *
*********************************************************************
*  READ A MESSAGE
*********************************************************************
         BAL   R8,RDINIT               READ  FROM SERIES/1
         BAL   R4,CHKRTN               CHECK RETURN CODE
*********************************************************************
*  WRITE A MESSAGE
*********************************************************************
WRIT     BAL   R8,WRINIT               WRITE TO SERIES/1
         BAL   R4,CHKRTN               CHECK RETURN CODE
         BAL   R8,RDACK                READ SERIES/1 ACKNOWLEDGEMENT
         BAL   R4,CHKRTN               CHECK RETURN CODE
*********************************************************************
*         ENDDO
*********************************************************************
         BCT   R3,LOOP                 IF R3 NE ZERO CONTINUE
*********************************************************************
*         OUTPUT 'SAMPLEC TEST ENDED'
*********************************************************************
         LA    R2,MSG04                PRINT END OF
         BAL   R5,PRINTIO              TEST MESSAGE.
*********************************************************************
*         ISSUE CLOSE MACRO CHANNEL ATTACH DCB
*********************************************************************
         MVC   COP(8),CLOSE            CURRENT OP = CLOSE
         CLOSE CADCB                   CLOSE LINE DCB
*********************************************************************
*         ISSUE CLOSE MACRO FOR PRINTER DATASET
*********************************************************************
         CLOSE PRINTDCB                CLOSE PRINTER DCB
EXIT     EQU   *
         L      SAVEREG,SAVEAREA+4     RESTORE SAVEREG
         RETURN (14,12),RC=0           EXIT
```

Figure 65 (Part 3 of 8). Host Sample Program

```
*********************************************************************
*   INTERNAL SUBROUTINE CHKRTN: CHECK I/O RETURN CODES
*         CALLING SEQUENCE:     BAL R4,CHKRTN
*********************************************************************
CHKRTN   EQU   *
         CLI   DECSDECB,X'7F'          IF ECB POST CODE IS BAD
         BNE   CHKCC41                   BRANCH
         CLI   DECFLAGS,X'00'          ELSE IF FLAGS NOT ZERO
         BNE   CHKFL1                          BRANCH
         BR    R4                            ELSE RETURN
CHKFL1   CLI   DECFLAGS,X'F0'          IF DEVICE HAS NOT BECOME READY
         BNE   CHKFL2                    BRANCH
         MVI   READY,X'FF'             ELSE SET READY FLAG
CHKFL2   BR    R4                      RETURN
CHKCC41  CLI   DECSDECB,X'41'          IF ECB POST CODE IS I/O ERROR
         BE    IOERROR                   BRANCH
         MVC   WORK(1),DECSDECB        ELSE LOAD POST CODE INTO WORK
         UNPK  MSG02+28(3),WORK(2)       LOAD CHAR INTO MSG
         MVI   MSG02+30,C' '             INSERT BLANK
         TR    MSG02+28(2),TABL-240      TRANSLATE
         LA    R2,MSG02                  PRINT
         BAL   R5,PRINTIO                      MESSAGE
         ABEND 4,DUMP                    ABEND
IOERROR  EQU   *
         TM    DECERRST,X'80'          IF NOT SIO ERROR
         BNO   IOERR                     BRANCH
         TM    DECCSWST,X'02'          ELSE IF NOT UNIT CHECK
         BNO   IOERR                     BRANCH
         TM    DECSENS0,X'40'          ELSE IF NOT INTERVENTION REQ'D
         BNO   IOERR                     BRANCH
         LA    R2,MSG10                ELSE
         BAL   R5,PRINTIO                PRINT INTERVENTION REQ'D
         B     EXIT                    EXIT
IOERR    MVC   WORK(1),DECFLAGS        LOAD FLAGS INTO WORK
         UNPK  MSG03+33(3),WORK(2)     LOAD CHAR INTO MSG
         MVI   MSG03+35,C' '             INSERT BLANK
         TR    MSG03+33(2),TABL-240    TRANSLATE
         LA    R2,MSG03                PRINT
         BAL   R5,PRINTIO                    MESSAGE.
         ABEND 5,DUMP                    ABEND
```

Figure 65 (Part 4 of 8). Host Sample Program

```
*********************************************************************
*   INTERNAL SUBROUTINE PRINTIO: PRINT MESSAGE
*        CALLING SEQUENCE:   LA  R2,MSGXX    ADDR OF MSG IN R2(NOTE)
*                            BAL PRINTIO,R5  PRINT MESSAGE
* NOTE:  FIRST BYTE PRECEDING THE MESSAGE HAS LENGTH OF MESSAGE.
*********************************************************************
PRINTIO  LR    R1,R2                  GET ADDR OF MSG
         BCTR  R1,0                   GET ADDR OF MSG LENGTH
         CLI   0(R1),128              IF MSG NOT TOO LONG
         BNH   PRINT                    BRANCH
         LA    R1,127                 ELSE
         B     PRINT1                   TRUNCATE TO MAX LENGTH
PRINT    IC    R1,0(R1)               GET MSG LENGTH
         BCTR  R1,0                   SUBTRACT ONE FOR MVC INSTR
PRINT1   EX    R1,MOVE                EXECUTE MOVE INSTRUCTION
         PUT   PRINTDCB,PRTBUF        PRINT IT
         SNAP  DCB=SNAPDCB,ID=10,
               STORAGE=(PRTBUF,EPRTBUF)
         MVI   PRTBUF,C' '            SET PRINT
         MVC   PRTBUF+1(127),PRTBUF           BUFFER TO BLANKS.
         BR    R5                     RETURN
*********************************************************************
*     INTERNAL SUBROUTINE EMSG: PRINT BTAM MACRO ERROR MESSAGE
*          CALLING SEQUENCE: B  EMSG   (OR EQUIVALENT)
*          ERROR CODE IN R15
*********************************************************************
EMSG     STC   R15,WORK               SAVE RC INTO WORK
         UNPK  MSG05+46(3),WORK(2)    CONVERT TO ZONED FORMAT
         MVI   MSG05+48,C' '          INSERT BLANK FOR SIGN POSITION
         TR    MSG05+46(2),TABL-240   TRANSLATE TO EBCDIC FOR PRINTING
         MVC   WORK(1),DECSDECB       MOVE DECSDECB INTO WORK
         UNPK  MSG05+59(3),WORK(2)    CONVERT TO ZONED FORMAT
         MVI   MSG05+61,C' '          INSERT BLANK FOR SIGN POSITION
         TR    MSG05+59(2),TABL-240   TRANSLATE TO EBCDIC FOR PRINTING
         MVC   MSG05+26(8),COP        INSERT CURRENT BTAM OPERATION
         LA    R2,MSG05               PRINT
         BAL   R5,PRINTIO             ERROR MESSAGE
         B     EXIT                   EXIT
```

Figure 65 (Part 5 of 8). Host Sample Program

```
*********************************************************************
*     BTAM I/O REQUESTS
*********************************************************************
* READ ACKNOWLEDGE MESSAGE
RDACK    LA    R2,MSG11                 RDACK MESSAGE
         B     RDALL                    PRINT MESSAGE
* READ INITIAL
RDINIT   LA    R2,MSG06                 RDINIT MESSAGE
RDALL    BAL   R5,PRINTIO               PRINT MESSAGE
         MVC   COP(8),READ              CURRENT OP = READ
         READ  (DECBREG),TI,CADCB,AREAIN,LIN,,RLN,MF=E
         B     WAIT
* WRITE INITIAL
WRINIT   LA    R2,MSG07                 WRINIT MESSAGE
         BAL   R5,PRINTIO               PRINT MESSAGE
         MVC   COP(8),WRITE             CURRENT OP = WRITE
         WRITE (DECBREG),TI,CADCB,AREAOUT,LOUT,,RLN,MF=E
         B     WAIT
* WRITE UNPROTECTED ERASE
WRUNER   LA    R2,MSG08                 WRUNER MESSAGE
         BAL   R5,PRINTIO               PRINT MESSAGE
         MVC   COP(8),WRITE             CURRENT OP = WRITE
         WRITE (DECBREG),TUS,CADCB,AREAOUT,1,,RLN,MF=E
         B     WAIT
*********************************************************************
*     WAIT FOR I/O COMPLETION
*********************************************************************
WAIT     LTR   R15,R15                  IF I/O RETURN CODE IS NONZERO
         BNZ   EMSG                       BRANCH
         WAIT  1,ECB=(DECBREG)          ELSE WAIT FOR I/O TO COMPLETE
         SNAP  DCB=SNAPDCB,ID=20,PDATA=(PSW,REGS),
               STORAGE=(SAVEAREA,PRINTDCB)
         CLC   COP(8),READ              IF COP EQ READ
         BE    WRUNER                     BRANCH TO WRITE ACK
         BR    R8                       RETURN
```

Figure 65 (Part 6 of 8). Host Sample Program

```
*********************************************************************
*     DATA DECLARATIONS
*********************************************************************
          DS    0F                          WORD ALIGNMENT
          LTORG
SAVEAREA  DS    18F                         SAVE AREA
LIMIT     DC    F'0'                        LIMIT COUNTER VALUE
MOVE      MVC   PRTBUF(1),0(2)              MOVE INSTR IS EXECUTED BY EX
WORK      DC    X'000F'                     WORK SPACE
          DC    AL1(L'MSG00)
MSG00     DC    C'SAMPLEC TEST STARTED'
          DC    AL1(L'MSG01)
MSG01     DC    C'CHANNEL ATTACH DEVICE OPENED.'
          DC    AL1(L'MSG02)
MSG02     DC    C'SAMPLEC:  COMPLETION CODE = XX    '
          DC    AL1(L'MSG03)
MSG03     DC    C'SAMPLEC:  I/O ERROR . DECFLAGS = XX  '
          DC    AL1(L'MSG04)
MSG04     DC    C'SAMPLEC TEST ENDED'
          DC    AL1(L'MSG05)
MSG05     DC    C'BTAM OPERATION ERROR FROM XXXXXXXX MACRO.
                RC= YY DECSDX ECB= ZZ '
          DC    AL1(L'MSG06)
MSG06     DC    C'RDINIT  '
          DC    AL1(L'MSG07)
MSG07     DC    C'WRINIT  '
          DC    AL1(L'MSG08)
MSG08     DC    C'WRUNER  '
          DC    AL1(L'MSG09)
MSG09     DC    C'CHANNEL ATTACH DEVICE COULD NOT BE OPENED.'
          DC    AL1(L'MSG10)
MSG10     DC    C'INTERVENTION REQUIRED ON CHANNEL ATTACH DEVICE.
                ISSUE X SERIES/1 OPERATOR COMMAND: "STDV CA1" '
          DC    AL1(L'MSG11)
MSG11     DC    C'RDACK   '
COP       DC    CL8' '                      CURRENT BTAM OPERATION
READ      DC    CL8'READ'                           READ
WRITE     DC    CL8'WRITE'                          WRITE
CLOSE     DC    CL8'CLOSE'                          CLOSE
```

Figure 65 (Part 7 of 8). Host Sample Program

```
OPEN     DC    CL8'OPEN'                      OPEN
         DS    0F                        FULLWORD ALIGNMENT
PRTBUF   DC    128C' '                   PRINT BUFFER
EPRTBUF  EQU   *-4                       ADDR OF LAST WORD IN PRTBUF
READY    DC    X'00'                     READY FLAG
TABL     DC    C'0123456789ABCDEF'       TRANSLATE TABLE
AREAIN   DC    256C' '                   INPUT AREA
EAREAIN  EQU   *-4                       ADDR OF LAST WORD IN INPUT
AREAOUT  DC    C'THE QUICK BROWN FOX '   AREA
         DC    C'JUMPED OVER THE LAZY'
         DC    C' DOG AND THE DISH RA'
         DC    C'N AWAY WITH THE SPOO'
         DC    C'N * $RIGX0004$ * () '
         DC    C'SYSTEM/370 -> SERIES'
         DC    C'/1      '
ENDOUT   EQU   *
NLIMIT   EQU   2                         LOOP COUNT
LIN      EQU   AREAOUT-AREAIN            LENGTH OF INPUT AREA
LOUT     EQU   ENDOUT-AREAOUT            LENGTH OF INPUT AREA
*****************************************************************
*     CONTROL BLOCKS
*****************************************************************
         READ  CADECB,TI,AREAIN,LIN,,RLN,MF=L
CADCB    DCB   DSORG=CX,MACRF=(R,W),DDNAME=S1GROUP,EROPT=E
PRINTDCB DCB   BLKSIZE=128,DDNAME=PRINTER,DSORG=PS,
               LRECL=128,MACRF=(PM),RECFM=FB
SNAPDCB  DCB   DSORG=PS,RECFM=VBA,MACRF=(W),BLKSIZE=1632,LRECL=125,
               DDNAME=SNAP
         DCBD  DSORG=BS,DEVD=DA
         IECTDECB
         END
```

Figure 65 (Part 8 of 8). Host Sample Program

Part 3. Specialized Series/1 Event Driven Executive Communications Methods

Part 3 discusses the two methods of communications that you can use on Series/1 only with the Event Driven Executive operating system:

- Series/1-to-Series/1 Attachment
- General Purpose Interface Bus (GPIB) Adapter.

Notes

Chapter 5. Series/1-to-Series/1 Attachment Support

Your Series/1 can communicate directly with the processor of another Series/1 in a configuration called a Series/1-to-Series/1 attachment. The two processors communicate over a connection established by RPQ D02241 and RPQ D02242 attachment cards plugged into each processor's I/O channel and connected by a cable. (One processor uses RPQ D02241 and the other RPQ D02242.) Each RPQ card connected to a given Series/1 must have its own cable.

For hardware installation and configuration details, refer to the *Series/1-to-Series/1 Attachment (RPQs D02241 and D02242) Custom Feature*.

Application programs running on both processors control the transfer of data between the two Series/1s. A separate synchronization program, also running simultaneously on the two processors, synchronizes the execution of the application programs.

To communicate with the processor that your Series/1 is attached to, you must write application programs to perform data transfers, and synchronization programs to control the application programs.

Planning the Series/1-to-Series/1 Application

Certain requirements and restrictions apply to the Series/1 to Series/1 application.

Planning the Series/1-to-Series/1 Application *(continued)*

Processor Relationships

Normally, the attached Series/1 systems communicate in a peer-to-peer relationship. Both processors contend with equal priority to initiate data transfers. However, the Series/1 to Series/1 attachment allows one processor to IPL the other (remote IPL). This establishes a primary-and-secondary processor relationship between the two Series/1s. Only the processor connected with RPQ D02241 can perform the remote IPL. When remote IPL is complete, the processor with RPQ D02241 functions as the primary and the processor with RPQ D02242 as the secondary. The primary processor controls the data transfers, thus eliminating contention between the two.

Initiating Data Transfers

Either Series/1 can initiate a data transfer to the other, regardless of the relationship between the processors. A data transfer can write data to the other processor, read data from it, or issue a control instruction. The receiving processor must always respond to a data transfer by sending back the opposite instruction to the initiating processor. For example, if the initiating processor performs a write operation, the receiving processor must answer with a read operation.

Responding to External Events

A processor that receives an external interrupt while attached to another Series/1 can record the external event by posting an event control block (ECB). For example, one processor may need to receive data from a device attached to it while it is in the middle of talking to the other processor.

However, it is possible to post an event control block even if the processor is not communicating with the other Series/1. This enables other terminal I/O tasks to execute on the posting processor while it waits for communications from the other processor.

Figure 66 on page CO-177 illustrates setup and usage of posting a user ECB. The program should execute on both processors.

The program has three tasks:

- Task 1 performs a write (PRINTEXT) and is driven by an external event, in this case task 3, which is a timer task. It could be sensor I/O or BSC as well.
- Task 2 performs a read (READTEXT). It responds only to an attention interrupt on the Series/1-to-Series/1 link, or to a post from task 1.
- Task 3 is a timer task, intended to simulate external stimuli.

```
T1        PROGRAM  GO,30
          PRINT    OFF
          COPY     CCBEQU
          PRINT    ON
          EJECT
GO        RESET    ECB                    RESET ECB FOR S1S1
          RESET    TECB                   RESET ECB FOR TIMER TASK
          MOVE     BUF-2,256              SET TO READ/WRITE 256 BYTES
          MOVE     BUF-4,BUF-2
          ENQT     S1S1                   GET S1S1
          TCBGET   #1,$TCBCCB             GET CCB ADDRESS OF S1S1
          MOVEA    #2,ECB                 SET ECB ADDR IN CCB
          MOVE     ($CCBS1EI+2,#1),#2,TKEY=0    DO IT
          TCBGET   #2,$TCBAKR             GET AKR
          AND      #2,X'0030'             AND TO GET OP2
          SHIFTR   #2,4                   MOVE IT OVER TO BITS 13-15
          MOVE     ($CCBS1EI,#1),#2,TKEY=0    SET INTO CCB
          DEQT     S1S1                   GIVE UP S1S1
          ATTACH   T2                     START TIMER TASK
          ATTACH   T3                     START READ TASK
*         TASK 2 ALWAYS DOES A WRITE (PRINTEXT)
WAITT1    WAIT     TECB                   WAIT FOR TIMER TASK
          RESET    TECB                   CLEAR TIMER ECB
          ENQT     S1S1
          PRINTEXT BUF,XLATE=NO           WRITE DATA
*          MOVE     T1STAT,T1             CHECK STATUS
         IF       (T1STAT,EQ,-1)         IF OK
           ADD    NUMT1,1,PREC=DS        BUMP COUNT
         ELSE
           ADD    NUMT1E,1,PREC=DS       IF ERROR, BUMP COUNT
*          DUE TO THE ASYNCHRONOUS NATURE OF THE S1/S1
*          WE MUST POST THE ECB WHEN WE GET AN ERROR HERE - ASSUMING
*          IT IS AN INVALID SEQUENCE ERROR (PRINTEXT FACING PRINTEXT)
           POST   ECB,-1                  POST THE ECB JUST IN CASE
          ENDIF
          DEQT
          GOTO     WAITT1                 LOOP
*         TASK 2 ALWAYS DOES A READ
T2        TASK     T2GO,25
T2GO      EQU      *
```

Figure 66 (Part 1 of 2). Program for Posting an Event Control Block

```
SLEEP    WAIT     ECB                    WAIT ON ECB POSTED BY S1S1
         RESET    ECB                    RESET ECB
         ENQT     S1S1
         MOVE     BUF,0,DWORD            CLEAR HEADER AREA
         TERMCTRL STATUS,BUF,WAIT=NO     SEE IF A READ
         IF       (BUF,EQ,ZERO,4)        IF NO OPERATION READY
           DEQT
           GOTO   SLEEP                  GO BACK TO SLEEP
         ENDIF
         AND      BUF,X'2000'            IF OPERATION READY
         IF       (BUF,EQ,ZERO)          SEE IF REQUIRES A READTEXT
           DEQT
           GOTO   SLEEP                  IF NOT, GO BACK TO SLEEP
         ENDIF
         READTEXT BUF,XLATE=NO           IF REQUIRES A READ, DO IT
         MOVE     T2STAT,T2              CHECK STATUS
         IF       (T2STAT,EQ,-1)
           ADD    NUMT2,1,PREC=DS        BUMP COUNT
         ELSE
           ADD    NUMT2E,1,PREC=DS       BUMP ERROR COUNT
         ENDIF
         DEQT
         GOTO     SLEEP                  LOOP
*        TASK 3 MERELY WAKES UP TASK 1 PERIODICALLY
T3       TASK     T3GO,510
T3GO     EQU      *
         STIMER   50,WAIT
         POST     TECB
         GOTO     T3GO
         EJECT
ECB      ECB      -1
TECB     ECB      -1
T1STAT   DATA     F'0'
T2STAT   DATA     F'0'
NUMT1    DATA     D'0'
NUMT1E   DATA     D'0'
NUMT2    DATA     D'0'
NUMT2E   DATA     D'0'
TYPE     DATA     F'0'
ZERO     DATA     D'0'
         DATA     X'0808'
S1S1     IOCB     S1S1,BUFFER=BUF
BUF      BUFFER   1024,BYTES
         ENDPROG
         END
```

Figure 66 (Part 2 of 2). Program for Posting an Event Control Block

Programming for Series/1-to-Series/1 Attachment

You must write application programs using a set of Event Driven Language instructions that implement Series/1-to-Series/1 communication. Your program can perform any type of data transfer (read, write or control operation). You can also write a program to synchronize the execution of the application program on both processors.

Event Driven Language Instruction Set

The following Event Driven Language instructions allow communication between the Series/1s. For descriptions and syntax of the instructions, refer to the *Language Reference*

Instruction	Explanation
DEQT	Releases the Series/1 previously enqueued with the ENQT instruction
ENQT	Acquires exclusive right to communicate with the enqueued Series/1
IOCB	Identifies the Series/1 as the enqueued processor
PRINTEXT	Writes a message to the enqueued Series/1
READTEXT	Reads a message from the enqueued Series/1
TERMCTRL	Performs control functions on the enqueued Series/1

Figure 67. EDL Instructions for Communication between Series/1s

Basic Programming Tasks

To communicate with the other Series/1 processor, your program must perform the following tasks:

- Gain exclusive ability to communicate with the other processor (enqueue with ENQT instruction)
- Identify the other processor (IOCB instruction)
- Write data to or read data from the other processor (PRINTEXT or READTEXT instruction)
- Perform control functions on the other processor (TERMCTRL instruction).

In addition, your program should provide for error detection and handling, and keep a count of the data transfers.

Enqueuing the Other Processor

Before your Series/1 can make data transfers to the other Series/1, you must gain exclusive right to communicate with that processor. The instruction that performs this function is ENQT. The other processor is treated as a terminal that your Series/1 enqueues. No other device or processor is able to send data to the other Series/1 while you have it enqueued. Because ENQT treats the other processor as a terminal, you must specify its symbolic identity. You must refer to the label of the IOCB instruction, which identifies the enqueued Series/1 in your program. The IOCB instruction is covered in the section that follows.

Identifying the Enqueued Processor

You must specify the identity of the processor that your Series/1 is going to communicate with. The IOCB instruction provides this information.

Give the IOCB instruction a label when you code it. You must refer to this label in the ENQT and DEQT instructions for the program.

A typical way to identify the Series/1 in IOCB is "S1S1."

Writing Data to the Enqueued Processor

To write data to the other Series/1, code a PRINTEXT instruction. Besides using PRINTEXT to simply initiate a write operation, code it in response to a READTEXT instruction issued by the other processor.

Reading Data from the Enqueued Processor

To read data from the other processor, code a READTEXT instruction. Besides using READTEXT to simply initiate a read operation, code it in response to a PRINTEXT issued by the other processor.

Performing Control Functions on the Enqueued Processor

You can perform certain control functions on the other processor by coding TERMCTRL instructions. The functions that apply to the Series/1-to-Series/1 Attachment are listed in the chart below.

Function	Explanation	What to Code
ABORT	Enqueued processor sends back a message to your Series/1 telling it to terminate the last function	TERMCTRL ABORT
IPL	Performs remote IPL of the enqueued processor	TERMCTRL IPL

Figure 68 (Part 1 of 2). TERMCTRL Functions for Series/1-to-Series/1 Communications

Function	Explanation	What to Code
RESET	Resets to the last ENQT issued	TERMCTRL RESET
STATUS	Obtains status of enqueued processor	TERMCTRL STATUS

Figure 68 (Part 2 of 2). TERMCTRL Functions for Series/1-to-Series/1 Communications

Do not attempt to use any TERMCTRL functions other than those specifically for Series/1-to-Series/1 attachment.

Programming Considerations

Certain requirements and restrictions apply to programming for the Series/1-to-Series/1 attachment.

- If a program that has issued an ENQT loads a program (via LOAD), the new program will have the Series/1-to-Series/1 as its default terminal. Therefore, a DEQT instruction should always be issued to the attachment before issuing the LOAD.
- Data to be transferred must start at an even address.
- Byte counts (data length) must be even.
- Return codes must be examined by the application. The return code is placed in the first word of the task control block (TCB). The return codes for the operations performed by the attachment are described in the *Messages and Codes* with the PRINTEXT/READTEXT instructions.
- Synchronization between the two processors is possible only by checking and responding to the return codes and by using TERMCTRL STATUS. A listing of the return codes can be obtained by including the copy code "COPY ERRORDEF" in your program.
- Do not use the CT command of the $TERMUT1 utility to reconfigure the Series/1-to-Series/1. The CT command parameters are not valid for the Series/1-to-Series/1 Attachment. Using $TERMUT1 to change attributes will cause unpredictable results.
- Byte counts must be equal on each processor. For example, if processor A issues a READTEXT for 50 bytes, then processor B must respond with a PRINTEXT for 50 bytes. However, if Event Driven Executive terminal I/O buffer management is utilized, this may be very difficult to achieve unless direct I/O is used. Direct I/O means that a buffer is specified in the application program by an IOCB statement and the IOCB is enqueued (ENQT) prior to issuing a PRINTEXT/READTEXT instruction. Data is transferred directly from the buffer instead of the buffer generated by the TERMINAL configuration statement. Refer to the *Event Driven Executive Language Programming Guide* for a description of the IOCB statement and for further information on direct I/O.

Your program should use direct I/O, with XLATE=NO, and should always transmit fixed-length records. Terminal I/O buffer management allows the number of bytes to vary for a PRINTEXT, but not for a READTEXT. READTEXT always reads the number of bytes specified by the TERMINAL statement LINSIZE parameter. PRINTEXT writes the number of bytes specified by the LENGTH parameter or the number of bytes in the message itself.

You may not be able to synchronize the byte counts unless direct I/O is used. If, however, direct I/O is not used and buffering of data with PRINTEXT/READTEXT instructions is performed, the results of using these instructions are shown in Figure 69. The "error" in the figure means that an invalid data length status is returned to the application.

Initiating Processor	Responding Processor	Status Returned to Responding Processor
READTEXT of x bytes	PRINTEXT of x bytes	OK
	of less than x bytes	OK
	of more than x bytes	error
PRINTEXT of x bytes	READTEXT of x bytes	OK
	of less than x bytes	error
	of more than x bytes	OK

Figure 69. Usage of READTEXT/PRINTEXT without Direct I/O

- Except for the device address, use identical TERMINAL statements for each processor. The results will be unpredictable if the TERMINAL statements differ.

- The IPL function of the $S1S1UT1 utility reads a specified nucleus from a disk and sends this nucleus to the other processor. The IPL function does not merely trigger the other processor to IPL from a disk or diskette. Therefore, the other processor does not require a disk or diskette. If the other processor does not have a disk or diskette, then the nucleus being sent must contain the supervisor and your application program. One processor cannot load a program on another processor.

- The Series/1-to-Series/1 Attachment supports only a subset of the terminal I/O instructions. Using instructions other than those described in the *Language Reference* for Series/1-to-Series/1 can cause unpredictable results. In addition, only the RESET, ABORT, STATUS, and IPL functions of the TERMCTRL instruction should be used.

- When a processor issues a PRINTEXT or READTEXT instruction (with or without direct I/O), control is not returned to the issuing program until the data transfer is complete (the other processor issues the opposite operation). Therefore, it is not necessary to perform request/acknowledge operations.

- TERMCTRL RESET is used for error recovery. This operation causes the attachment to reset. The two applications must be synchronized to avoid an error/reset loop.

For example, an error/reset loop can occur if program A issues a PRINTEXT and encounters an error, issues a TERMCTRL RESET, and reissues the PRINTEXT. Program B, on the other hand, issues a READTEXT (at approximately the same time as program A's initial PRINTEXT) and receives an error. However, because of heavy processor or I/O activity, program B is unable to issue its TERMCTRL RESET until after program A has already performed its reset and reissued a PRINTEXT. Hence, program B's reset will cause program A to receive an error while program A is issuing its second PRINTEXT (retry). Both programs enter the loop as a result of trying to read or write while the programs are no longer synchronized and the attachments are being reset. Issuing an STIMER instruction (with WAIT) in the error recovery routine in program A can help avoid entering an error/reset loop.

- Only two of the possible errors can occur concurrently on both processors. These errors should be taken into consideration in your error recovery routine:
  - 1004 – Checksum error
  - 1008 – Time-out error.

Programming Examples

The following programs perform data transfers controlled by a synchronization program. The processors are assumed to be operating in a primary-to-secondary relationship. The PRIMARY program runs on the Series/1 using RPQ D02241; the SECOND program runs on the Series/1 using RPQ D02242. Both of these programs run simultaneously on the two processors. The SYNC program controls their execution.

Primary Processor Sample Program

The PRIMARY program controls the SECOND program. The processor running PRIMARY can abort the SECOND program. The operator presses the attention key and enters "AB" at the terminal that loaded PRIMARY. The command "MS" sends a message, a count of the transfers, and their length. Note the code for error recovery.

```
PRIMARY   PROGRAM      GO
*
*
          ATTNLIST     (AB,ABT,MS,MSG)
ABT       EQU          *
          MOVE         ABSW,1                          SET ABORT FLAG
          ENDATTN
MSG       EQU          *
          MOVE         MSW,1                           SET MESSAGE FLAG
          ENDATTN
GO        PRINTEXT     'ØI AM THE PRIMARY SERIES/1Ø'
          PRINTEXT     'Ø"> AB" TO ABORT SECONDØ'
          PRINTEXT     'Ø"> MS" TO SEND MESSAGE FLASHØ'
GETSIZE   EQU          *
          GETVALUE     BUFI,'ØENTER DATA LENGTH(6 ==> 4096)Ø'
          IF           (BUFI,LT,6),OR,(BUFI,GT,4096),GOTO,GETSIZE
          AND          BUFI,X'0001',RESULT=TEMP
          IF           TEMP,NE,0
          PRINTEXT     'ØDATA LENGTH CANNOT BE ODDØ'
          GOTO         GETSIZE
          ENDIF
          QUESTION    'ØRESET S1S1? ',NO=GO1
          ENQT        S1S1
          TERMCTRL    RESET
          DEQT
GO1       EQU          *
          QUESTION    'ØREADY? ',YES=GO2
          QUESTION    'ØTERMINATE? ',YES=STOP1,NO=GO
GO2       EQU          *                               SET UP CONSTANTS
*                                                      FOR RESTART
          MOVE         BUFI+2,BUFI                     SET DATA LENGTH
          MOVE         STAT,0                          CLEAR STATUS
          MOVE         ABSW,0                          CLEAR ABORT SWITCH
          MOVE         MSW,0                           CLEAR MESSAGE SWITCH
          MOVE         TYPE,-1                         SET TYPE
          MOVE         COUNT,0,DWORD                   CLEAR COUNTER
          MOVE         HBUF,0,(6,BYTES)                CLEAR S1S1 HEADER
          MOVE         CSS,0,(24,BYTES)                CLEAR S1S1 CSS
          ENQT         S1S1
```

Figure 70 (Part 1 of 4). Primary Processor Sample Program

```
CONT      EQU       *
          MOVE      TYPE,1                    SET TYPE = WRITE
          ADD       COUNT,1,PREC=D            INCREMENT COUNTER
          MOVE      BUF,COUNT,DWORD          MOVE COUNTER TO BUFFER
          MOVE      BUF+4,MSW                 MOVE MESSAGE SWITCH
*                                             TO BUFFER
          MOVE      MSW,0                     CLEAR MESSAGE SWITCH
          PRINTEXT  BUF,XLATE=NO
          MOVE      STAT,PRIMARY              GET STATUS
          IF        STAT,NE,-1                CHECK FOR BAD STATUS
          CALL      S1ERR                    CALL ERROR SUBROUTINE
          GOTO      ERROR                    ERROR EXIT
          ENDIF
          MOVE      BUF,0,(6,BYTES)           CLEAR BUFFER
          MOVE      TYPE,0                    SET TYPE = READ
          IF        (ABSW,NE,0),GOTO,ABORT    CHECK OPERATOR
*                                             ABORT REQUEST
          ADD       COUNT,1,PREC=D            INCREMENT COUNTER
          READTEXT  BUF,XLATE=NO              ACKNOWLEDGE
          MOVE      STAT,PRIMARY              GET STATUS
          IF        STAT,NE,-1                IF ERROR -
          CALL      S1ERR                    CALL ERROR SUBROUTINE
          GOTO       ERROR
          ENDIF
          IF        (BUF,NE,COUNT,DWORD)       IF COUNT BAD -
*                                              OUT OF SYNC
          PRINTEXT  '@OUT OF SYNC - COUNT FAILURE@'
          CALL      S1ERR
          GOTO      ERROR
          ENDIF
          IF        BUF+4,NE,0                CHECK MESSAGE
*                                             FLAG REQUEST
```

Figure 70 (Part 2 of 4). Primary Processor Sample Program

```
          DEQT
          PRINTEXT   '@MESSAGE RECEIVED FROM SECOND@'
          PRINTEXT   'COUNT= '
          PRINTNUM   COUNT,FORMAT=(12,0,I),TYPE=D
          PRINTEXT   '@DATA LENGTH = '
          PRINTNUM   BUFI,FORMAT=(6,0,I)
          PRINTEXT   '@'
          ENQT       S1S1
          ENDIF
          GOTO       CONT
*
*     ERROR EXIT
*     IF GET AN ERROR - RESET THE OTHER
*     DEVICE, WAIT, THEN RETRY.
*
ERROR     EQU        *
          DEQT
          ENQT       S1S1
          TERMCTRL   RESET                        RESET S1S1
          STIMER     200,WAIT                     WAIT 200 MS
          DEQT
          PRINTEXT   '@AUTO RESTART@'
          GOTO       GO2                          RESTART
*                                                 AUTOMATICALLY
STOP       EQU       *
           DEQT
           QUESTION  '@RESTART? ',YES=GO
STOP1      EQU       *
           PROGSTOP
*
*     ABORT ROUTINE
*
ABORT      EQU       *
           DEQT
           PRINTEXT  '@OPERATOR REQUESTED ABORT@'
ABORT1     EQU       *
           ENQT      S1S1
           STIMER    1000,WAIT                     WAIT 1 SECOND
           TERMCTRL  STATUS,HBUF                   SECONDARY TALKING TO ME?
           IF        (HBUF,NE,D0,DWORD)
           TERMCTRL  ABORT                      IF YES - ABORT SECOND
           DEQT
           PRINTEXT  '@ABORT ISSUED AS PER REQUEST@'
           GOTO      STOP                          END IT
           ENDIF
           DEQT
           PRINTEXT  '@SECOND DID NOT REQUEST DATA IN 1 SECOND@'
           GOTO      STOP                          END IT
```

Figure 70 (Part 3 of 4). Primary Processor Sample Program

```
*
*      S1S1 ERROR SUBROUTINE
*
            SUBROUT   S1ERR
            TERMCTRL  STATUS,HBUF,CSS
            DEQT
            ENQT
            PRINTEXT  '@BUF: '
            PRINTNUM  BUF
            PRINTEXT  '@TYPE: '
            PRINTNUM  TYPE
            PRINTEXT  '@COUNT: '
            PRINTNUM  COUNT,FORMAT=(12,0,I),TYPE=D
            PRINTEXT '@I/O ERROR ON S/1-S/1 - STATUS = '
            PRINTNUM  STAT,MODE=HEX
            PRINTEXT  ' HEX; '
            PRINTNUM  STAT
            PRINTEXT  ' DEC@'
            PRINTEXT  '@HEADER: '
            PRINTNUM   HBUF,2,MODE=HEX
            PRINTEXT  '@'
            PRINTEXT  'DIAGNOSTIC JUMPER WORD: '
            PRINTNUM  CSS,1,MODE=HEX
            PRINTEXT  '@'
            PRINTEXT  'CYCLE STEAL STATUS: '
            PRINTNUM  CSS+2,11,MODE=HEX
            PRINTEXT  '@'
            DEQT
            RETURN
HBUF        DATA      2F'0'                        S1S1 HEADER
CSS         DATA      12F'0'                       JUMPER + CYCLE
*                                                  STEAL STATUS
STAT        DATA      F'0'                         STATUS RETURN
TYPE        DATA      F'-1'                        READ/WRITE TYPE
COUNT       DATA      D'0'                         SYNC COUNTER
MSW         DATA      F'0'                         MESSAGE SWITCH
ABSW        DATA      F'0'                         ABORT SWITCH
S1S1        IOCB      S1S1,BUFFER=BUF
BUF         BUFFER    4096,BYTES,INDEX=BUFI
D0          DATA      D'0'                         DWORD CONSTANT
TEMP        DATA      F'0'                         TEMP STORAGE
            ENDPROG
            END
```

Figure 70 (Part 4 of 4). Primary Processor Sample Program

Secondary Processor Sample Program

The SECOND program is controlled by the PRIMARY program. The attention command "MS" sends a message, transfer count, and data length to PRIMARY.

```
SECOND   PROGRAM      GO
*
*
         ATTNLIST     (MS,MSG)
MSG      EQU          *
         MOVE         MSW,1                        SET MESSAGE SWITCH
         ENDATTN
GO       PRINTEXT     '@I AM THE SECOND@'
         PRINTEXT     '@"> MS" TO SEND MESSAGE FLASH@'
         QUESTION     '@RESET S1S1? ',NO=GO1
         ENQT         S1S1
         TERMCTRL     RESET                        RESET DEVICE
         DEQT
GO1      EQU          *
         QUESTION     '@READY? ',YES=GO2
         QUESTION     '@TERMINATE? ',YES=STOP1,NO=GO
GO2      EQU          *
*
*      HERE WE WAIT FOR THE MASTER TO INITIATE A DATA
*      TRANSFER. WHEN IT DOES, WE USE ITS DATA LENGTH
*      AND PROCEED.
*
           STIMER     1000,WAIT                    WAIT 1 SECOND
           ENQT       S1S1
           TERMCTRL   STATUS,HBUF                  GET STATUS
           DEQT
           IF         (HBUF,EQ,D0,DWORD)           IF NO DATA HERE YET
             DEQT
             PRINTEXT '@NO DATA FROM MASTER IN 1 SECOND@'
             GOTO     GO2                          SHOULD HAVE HAD
*                                                 DATA BY NOW
           ENDIF
           MOVE       BUFI,HBUF+2                   GET DATA LENGTH
           MOVE       BUFI+2,HBUF+2                 INTO BUFFER
           MOVE       STAT,0                        CLEAR STATUS
           MOVE       COUNT,0,DWORD                 CLEAR COUNTER
           MOVE       TYPE,-1                       INIT TYPE
           MOVE       HBUF,0,(6,BYTES)              CLEAR S1S1 HEADER
           MOVE       CSS,0,(24,BYTES)              CLEAR CSS
           ENQT       S1S1
```

Figure 71 (Part 1 of 4). Secondary Processor Sample Program

```
CONT    EQU         *
        MOVE        TYPE,0                      SET TYPE = READ
        ADD         COUNT,1,PREC=D              INCREMENT COUNTER
        MOVE        BUF,0,(6,BYTES)             CLEAR BUFFER
        READTEXT    BUF,XLATE=NO
        MOVE        STAT,SECOND                 GET STATUS
        IF          STAT,NE,-1
        CALL        S1ERR                   ERROR - CALL
*                                         ERROR SUBROUT
        GOTO        ERROR
        ENDIF
        IF          (BUF,NE,COUNT,DWORD)        CHECK COUNT FOR SYNC
        DEQT
        PRINTEXT    '@OUT OF SYNC - COUNT FAILURE@'
        CALL        S1ERR
        GOTO        ERROR
        ENDIF
        IF          (BUF+4,NE,0)                CHECK FOR MESSAGE
*                                               REQUEST
        DEQT
        PRINTEXT    '@MESSAGE RECEIVED FROM MASTER@'
        PRINTEXT    'COUNT = '
        PRINTNUM    COUNT,FORMAT=(12,0,I),TYPE=D
        PRINTEXT    '@DATA LENGTH = '
        PRINTNUM    BUFI,FORMAT=(6,0,I)
        PRINTEXT    '@'
        ENQT        S1S1
        ENDIF
```

Figure 71 (Part 2 of 4). Secondary Processor Sample Program

```
         ADD          COUNT,1,PREC=D                INCREMENT COUNTER
         MOVE         BUF,COUNT,DWORD               MOVE COUNT TO BUFFER
         MOVE         BUF+4,MSW                     MOVE MESSAGE SWITCH
*                                                  TO BUFFER
         MOVE         MSW,0                         CLEAR MESSAGE SWITCH
         MOVE         TYPE,1                        SET TYPE = WRITE
         PRINTEXT     BUF,XLATE=NO
         MOVE         STAT,SECOND                   GET STATUS
         IF           STAT,NE,-1                    IF ERROR - CALL
*                                                    S1S1 ERROR
         CALL         S1ERR
         GOTO         ERROR                      GO TO ERROR ROUTINE
         ENDIF
         GOTO         CONT                          CONTINUE
STOP     EQU          *
         DEQT
         QUESTION     '@RESTART? ',YES=GO
STOP1    EQU          *
         PROGSTOP
*
*      IF THE ERROR IS AN ABORT REQUEST, TERMINATE THE
*      EXERCISE. IF IT IS ANY OTHER ERROR, RESET THE
*      DEVICE, AND START OVER.
*
ERROR    EQU          *
         DEQT
         IF           (STAT,EQ,1010)                SEE IF ABORT
         PRINTEXT     '@MASTER ISSUED ABORT@'
         GOTO         STOP
         ENDIF
         ENQT         S1S1
         TERMCTRL     RESET                        RESET DEVICE
         DEQT
         GOTO         GO2
```

Figure 71 (Part 3 of 4). Secondary Processor Sample Program

```
*
*       S1S1        ERROR SUBROUTINE
*
        SUBROUT     S1ERR
        TERMCTRL    STATUS,HBUF,CSS
        DEQT
        ENQT
        PRINTEXT    '@BUF: '
        PRINTNUM    BUF
        PRINTEXT    '@TYPE: '
        PRINTNUM    TYPE
        PRINTEXT    '@COUNT: '
        PRINTNUM    COUNT,FORMAT=(12,0,I),TYPE=D
        PRINTEXT    '@I/O ERROR ON S/1-S/1 - STATUS = '
        PRINTNUM    STAT,MODE=HEX
        PRINTEXT    ' HEX; '
        PRINTNUM    STAT
        PRINTEXT    ' DEC@'
        PRINTEXT    '@HEADER: '
        PRINTNUM    HBUF,2,MODE=HEX
        PRINTEXT    '@'
        PRINTEXT    'DIAGNOSTIC JUMPER WORD: '
        PRINTNUM    CSS,1,MODE=HEX
        PRINTEXT    '@'
        PRINTEXT    'CYCLE STEAL STATUS: '
        PRINTNUM    CSS+2,11,MODE=HEX
        PRINTEXT    '@'
        DEQT
        RETURN
S1S1    IOCB        S1S1,BUFFER=BUF
HBUF    DATA        2F'0'                       S1S1 HEADER BUFFER
CSS     DATA        12F'0'                      JUMPER + CYCLE
*                                               STEAL STATUS
STAT    DATA        F'0'                        S1S1 STATUS
TYPE    DATA        F'-1'                       READ/WRITE TYPE
COUNT   DATA        D'0'                        SYNC COUNTER
MSW     DATA        F'0'                        MESSAGE SWITCH
ABSW    DATA        F'0'                        ABORT SWITCH
BUF     BUFFER      4096,BYTES,INDEX=BUFI
D0      DATA        D'0'                        DWORD CONSTANT
        ENDPROG
        END
```

Figure 71 (Part 4 of 4). Secondary Processor Sample Program

Synchronization Program (SYNC)

The synchronization program shows a way to synchronize operations between two processors. Neither processor begins communicating until some external event sets FLAG. When FLAG is set, the program responds as follows:

- FLAG = 0 indicates no external event.
- FLAG = 1 indicates write to other processor.
- FLAG = 2 indicates read from other processor.
- FLAG = 3 is the termination indicator.

This program runs on both processors.

```
SYNC   PROGRAM     A
*
       ATTNLIST    (WR,F1,RD,F2,EN,END)
F1     MOVE        FLAG,1                    EXTERNAL EVENT= WRITE
       ENDATTN
F2     MOVE        FLAG,2                    EXTERNAL EVENT=READ
       ENDATTN
END    MOVE        FLAG,3                    EXTERNAL EVENT=STOP PGM
       ENDATTN
A      MOVE        BUFX,1024                 SET BUFFER FOR WRITES
A1     ENQT        S1S1                      GET THE S1S1
       TERMCTRL    STATUS,HBUF,WAIT=NO       OBTAIN S1S1 STATUS
       DEQT
       IF          (HBUF+2,EQ,0)             IF NO ACTION YET
         STIMER    5000,WAIT                 WAIT 5 SECONDS
         IF        (FLAG,EQ,0)               EXTERNAL EVENT NOT
*                                            OCCURRED YET
          DEQT
          PRINTEXT '@NOTHING HAS OCCURRED YET IN 5 SECONDS@'
          GOTO      A1
         ENDIF
         IF        (FLAG,EQ,3),GOTO,STOP     EXTERNAL EVENT = TERMINATE
         IF        FLAG,EQ,1                 EXTERNAL EVENT = WRITE
          DEQT
          PRINTEXT '@WRITING TO OTHER CPU@'
          ENQT     S1S1                        GET S1S1
           PRINTEXT BUF,XLATE=NO            WRITE TO OTHER CPU
           DEQT
           MOVE    FLAG,0                    CLEAR FLAG TO IDLE
           GOTO    A1                         CONTINUE
         ENDIF
```

Figure 72 (Part 1 of 2). Synchronization Sample Program

```
        IF        FLAG,EQ,2                  EXTERNAL EVENT = READ
         DEQT
         PRINTEXT '@READING FROM OTHER CPU@'
         ENQT     S1S1                       GET S1S1
         READTEXT BUF,XLATE=NO              READ FROM OTHER CPU
         DEQT
          MOVE    FLAG,0                     CLEAR FLAG TO IDLE
          GOTO    A1                         CONTINUE
        ENDIF
      ELSE
       SHIFTL    HBUF,2,1,RESULT=TEMP       SEE IF READ OR
*                                            WRITE REQUEST
        IF        TEMP,LT,0                  IT DID A WRITE
         DEQT
         PRINTEXT '@READING WHAT IT SAID@'
         ENQT     S1S1
         READTEXT BUF,XLATE=NO          READ WHAT IT WROTE
         DEQT
         GOTO    A1                         CONTINUE
        ELSE
         DEQT
         PRINTEXT '@WRITING WHAT IT SAID@'
         ENQT     S1S1
         PRINTEXT BUF,XLATE=NO          WRITE WHAT IT WROTE
         DEQT
         GOTO     A1                         CONTINUE
        ENDIF
      ENDIF
      GOTO        A1                         CONTINUE
STOP  PROGSTOP                          END THE PROGRAM
S1S1  IOCB        S1S1,BUFFER=BUF
BUF   BUFFER      1024,BYTE,INDEX=BUFX       DATA BUFFER
FLAG  DATA        F'0'                       EXTERNAL EVENT FLAG
TEMP  DATA        F'0'
HBUF  DATA        2F'0'                      S1S1 HEADER
      ENDPROG
      END
```

Figure 72 (Part 2 of 2). Synchronization Sample Program

Series/1-to-Series/1 Attachment Support

Interacting with the Series/1-to-Series/1 Attachment (Using $S1S1UT1)

The $S1S1UT1 utility allows you to perform several functions on the other processor. These include remote IPL, data transfers and status operations. The utility also verifies that the attachment is installed correctly, and checks out an application program. You must always specify the identity of the other Series/1 the $S1S1UT1 commands:

- AB - Abort
- DD - Define device name
- IP - IPL the other processor
- RE - Read data
- RS - Reset device
- WR - Write data
- EN - End the program
- EC - Verify the Series/1 - Series/1 attachment
- ST - Obtain status.

You must have both attached processors actively communicating with the RE (read), WR (write), and AB (abort) commands. For example, if you issue a WR command on one processor, then you must issues a corresponding RE command on the other processor.

When you load $S1S1UT1, it immediately issues a DD command to obtain the ID of the Series/1 that loaded it.

Invoking $S1S1UT1

You invoke the $S1S1UT1 utility with the $L operator command or option 4.8 of the session manager. $S1S1UT1 must be active on two connected processors for processor-to-processor communication via the RE (read), WR (write), and AB (abort) commands. For example, if you issue a WR command on one processor, then you must issue a corresponding RE command on the other processor.

Aborting an Operation (AB)

The AB command causes the other processor to abort a pending operation on the initiating processor. The AB command is sent to the Series/1 specified in the most recent DD command. This command abnormally ends a data transfer operation.

Interacting with the Series/1-to-Series/1 Attachment (Using $S1S1UT1) *(continued)*

Example: AB command

```
COMMAND(?):  AB
ABORT OPERATION?  Y

COMMAND(?):
```

Specifying the Other Processor (DD)

For every function you perform with $S1S1UT1, you must identify the other Series/1. The specified processor receives all the issued subsequent utility commands until another DD is issued. The Series/1 name is the name specified on the TERMINAL configuration statement at system generation.

Example: DD command

```
COMMAND(?):  DD
S1S1 DEVICE NAME:  S1S100

COMMAND(?):
```

Verifying the Series/1-to-Series/1 Attachment (EC)

The EC command verifies that the attachment is installed correctly. It results in a continuous exchange of 1024-byte records between the attached processors. To terminate EC, press the attention key and enter EC.

Example: EC command

```
COMMAND(?):  EC
ECHO EXERCISE
ATTN ¶¶EC◀◀ TO STOP ECHO TEST
ATTEMPTING TO SYNCH UP
1K DATA TRANSMITTED
1K DATA RECEIVED
DATA CHECKS OUT!

(Attention)
> EC

COMMAND(?):
```

Interacting with the Series/1-to-Series/1 Attachment (Using $S1S1UT1) *(continued)*

IPL the Other Processor (IP)

You can issue the IP command only if your Series/1 is the primary. It issues an IPL (initial program load) command to the secondary processor. You must supply the member name and volume that contains the nucleus to be transferred to the secondary.

The nucleus being transferred must begin with the characters $EDXNUC. The IPL bootstrap program, IPLS1S1, must be located in the IPL volume; you are prompted to verify that the volume name specified is correct. You are also asked if the secondary has a disk or diskette device and for the address of that device. The specified disk or diskette becomes the default direct access device for disk I/O operations on the processor being initialized. The bootstrap program is sent to the secondary and it reads the specified nucleus, 1024 bytes at a time, across the attachment. Control is passed to the nucleus upon completion of the transfer.

Example: IP command

```
COMMAND(?):  IP

ENTER NUCLEUS DSN:  $EDXNUC

ENTER NULCEUS VOLUME:  EDX002
IS THERE A DEFAULT DISK DEVICE FOR THE NUCLEUS?  Y
SUPPLY DISK/DISKETTE DEVICE ADDRESS(HEX):  48
IPL PROGRAM NAME:  IPLS1S1  ON VOLUME:  EDX002
OK?  Y
READY FOR IPL?  Y
```

Reading Data from the Other Processor (RE)

The RE command issues an ENQT instruction for the terminal name specified by the most recent DD command. $S1S1UT1 then issues a READTEXT instruction for that terminal to read data from the other processor. If $S1S1UT1 is active on the other processor, a WR command must be issued to complete an RE command.

Example: RE command

```
COMMAND(?):  RE

MESSAGE RECEIVED!

THIS IS TEST DATA RECEIVED

COMMAND(?):
```

Interacting with the Series/1-to-Series/1 Attachment (Using $S1S1UT1) *(continued)*

Resetting Device (RS)

The RS command resets the to the Series/1 attachment specified by the most recent DD command. This command clears any pending interrupt or busy condition.

Example: RS command

```
COMMAND(?):  RS
RESET ATTACHMENT?  Y

COMMAND(?):
```

Obtaining Status of an Operation (ST)

The ST command obtains status information on an operation. The status returned by this command is the same as the information returned when a TERMCTRL STATUS instruction is issued.

Example: ST command

```
COMMAND(?):  ST
OBTAIN CYCLE STEAL STATUS ALSO?  Y
WAIT FOR HEADER?  N
HEADER WORDS:  1100 0400
READ(READTEXT) ISSUED BY OTHER CPU
NO. BYTES = 0400
DIAGNOSTIC JUMPER WORD:  02E6
CYCLE STEAL STATUS:        (11 words of status)

COMMAND(?):
```

Interacting with the Series/1-to-Series/1 Attachment (Using $S1S1UT1) *(continued)*

Writing Data to the Other Processor (WR)

The WR command issues an ENQT instruction to the Series/1 specified by the most recent DD command. A PRINTEXT instruction is then issued for that terminal to write data to the other processor. If $S1S1UT1 is active on the other processor, an RE command must be issued to complete a WR command.

Example: WR command

```
COMMAND(?):  WR
ENTER TEXT:
TEST DATA TO WRITE

MESSAGE SENT!

COMMAND(?):
```

Ending the Utility (EN)

The EN command ends the $S1S1UT1 utility.

Example: EN command

```
COMMAND(?):  EN

$S1S1UT1 ENDED AT 00:00:00
```

Chapter 6. General Purpose Interface Bus - IEEE Standard 488-1975

The General Purpose Interface Bus (GPIB) Adapter, when connected to the Series/1, allows it to communicate with up to 14 peripheral devices. The devices that GPIB can connect to the Series/1 include printers, plotters, graphics display units, and programmable laboratory equipment such as digital voltmeters, frequency analyzers, and signal generators.

The connection between the Series/1 and the GPIB devices is much like a multipoint connection. The Series/1 acts as the control station, and the GPIB devices as its tributaries.

You must write application programs to control communications between the Series/1 and the GPIB devices. Your program must perform polling and selection, and initiate all data transfer operations.

Planning for the GPIB Application

System Generation for GPIB

During system generation for the Series/1, you must include a TERMINAL statement in the configuration module ($EDXDEF) for each GPIB adapter. Also, you must include the GPIB device handler (IOSGPIB) in the supervisor link control data set ($LNKCNTL). Refer to the *Installation and System Generation Guide* for additional information on system generation.

You may want to connect GPIB to a particular partition so that SRQ interrupts adapter can be properly handled. The partition to which the GPIB adapter is initially connected is defined by the PART parameter of the TERMINAL statement. To modify the size of the system buffer generated for the GPIB adapter, use the LINSIZE parameter.

The following TERMINAL statement connects a GPIB adapter named GPIB to partition 2 and defines the system buffer to be 200 bytes in length.

```
GPIB    TERMINAL DEVICE=GPIB,ADDRESS=32,LINSIZE=200,PART=2
```

Relationship between Series/1 and GPIB Devices

The Series/1 always acts as the control station and the GPIB devices act as its tributary stations. A GPIB device can function in two roles:

- As a *talker*, a device is sending data to the Series/1 or to another device.
- As a *listener*, a device is receiving data from the Series/1 or another device.

The Series/1 controls communications so that only one device is talking at a time. However, any number of devices can be listening at the same time. The devices can function in either role. A device can act as talker during one data transfer, and as listener during another transfer. The Series/1 tells the devices which role to assume during each transfer operation.

The Series/1 must make the assignments of talker and listener(s) before a data transfer operation occurs. For example, to send data to a device, the Series/1 designates itself as the talker and the receiving device as the listener. The Series/1 then sends the data. To read the response from the device, the Series/1 designates the device as talker, and itself as listener. The Series/1 then reads the data.

Assigning Device Addresses

Each device on a GPIB bus has two addresses: one identifies the device as a listener and the other identifies it as a talker. Each listen address has a corresponding talk address, and vice versa. For example, if a device listens at address C'3', it talks at address C'S'.

The chart below shows the sets of listen and talk addresses; they are ASCII characters. Although GPIB device addresses are usually expressed as ASCII characters, they can also be coded in hexadecimal.

The Series/1 listen and talk addresses are C'5' and C'U' respectively; they cannot be changed.

Listen Address	Talk Address
(space)	@
!	A
&sqd.	B
#	C
$	D
%	E
&	F
'	G
(	H
)	I
*	J
+	K
,	L
—	M
.	N
/	O
0	P
1	Q
2	R
3	S
4	T
5	U
6	V
7	W
8	X
9	Y
:	Z
;	[
<	\
=	]
>	(caret)

Figure 73. Listen and Talk Addresses for GPIB Devices

Initializing and Configuring the Bus

You must initialize and configure the bus before the Series/1 and the GPIB device can communicate. These functions are performed by the TERMCTRL instruction, as described in the Programming section. The initialization consists of these operations:

1. Reset the adapter.

2. Clear the interface between Series/1 and GPIB; this makes the entire bus inactive.

3. Perform remote enable of GPIB; this enables all the devices attached to the bus.

Code these operations at the beginning of your application program. Unless an error occurs, you do not have to repeat the initialization.

Configuring the bus consists of assigning the roles of talker and listener(s) for the tasks in a program. It also allows the Series/1 to send data to listening devices.

Interrupt Handling

A service request (SRQ) is a device-generated signal that informs the controller (Series/1) that the device needs service. A program can be notified of the occurrence of an SRQ by coding a $PF255 reference in an attention list (ATTNLIST), with SCOPE=GLOBAL specified. Since the attention list receives control enqueued to the terminal, the main task (which is enqueued to GPIB) must issue a DEQT to GPIB. The following example shows an event control block (ECB) being posted when an SRQ occurs.

```
            ATTNLIST ($PF255,GOTSRQ),SCOPE=GLOBAL
            ..
            ..
            ..
*  ATTENTION LIST TASK THAT RECEIVES CONTROL
*  WHEN AN SRQ IS RECEIVED.
*
GOTSRQ      EQU      *
            POST     SRQECB          POST SRQ OCCURRENCE
            ENDATTN                  END ATTENTION PROCESSING
            ..
            ..
            ..
*  MAIN TASK
*  AT THIS POINT, INSTRUCTIONS HAVE BEEN ISSUED THAT
*  CAUSE A GPIB DEVICE TO ISSUE AN SRQ.
*
            DEQT                     DISCONNECT FROM GPIB
            WAIT     SRQECB          PROGRAM WAITS FOR EVENT
*                                    INDICATING SRQ OCCURRENCE
            ..
            ..
            ..
SRQECB      ECB                      SRQ EVENT CONTROL BLOCK
            ..
            ..
            ..
```

Note: A program that uses an attention list to receive SRQ interrupts in this manner must run in the partition to which the GPIB terminal is connected. This is initially defined at system generation time, but can be changed using the change partition (CP) command of $GPIBUT1.

A program can also handle an SRQ interrupt via the WAIT KEY instruction. The program remains in a wait state until an SRQ interrupt is received by the Series/1. However, as for all WAIT KEY instructions, an interrupt that occurs before the WAIT KEY is executed will be lost.

An SRQ does not identify the interrupting device. To determine this, the application program must poll all devices connected to the bus; see "Polling the Devices" on page CO-209.

Programming for the GPIB Application

You must write programs that allow communication between Series/1 and the GPIB devices. Your program must perform the following tasks:

- Initialize and configure the bus
- Control data transfer operations
- Poll and select devices
- Send data to devices
- Receive data from devices
- Perform control functions on devices.

Your programs must consist of Event Driven Language instructions. The instructions that apply specifically to the GPIB application are discussed in the next section.

Event Driven Language Instruction Set

Your program must contain the following EDL instructions, which perform the task of communicating between the Series/1 and the GPIB devices.

Instruction	Function
DEQT	Releases the device that Series/1 is communicating with
ENQT	Gives Series/1 exclusive right to communicate with a device
IOCB	Identifies the particular GPIB adapter that connects the device Series/1 wants to communicate with
PRINTEXT	Sends data to a device
READTEXT	Receives data from a device
TERMCTRL	Performs control functions on a device

For description and syntax of the instructions refer to the *Language Reference.*

Programming Considerations

Certain requirements and restrictions apply to programming for the GPIB application.

Performing Control Function (Using TERMCTRL)

Use only the TERMCTRL operands specific to GPIB. These are described in the *Language Reference*.

Forcing Output of Data

You must "force" the output of data to a GPIB device. If the Series/1 issues a PRINTEXT instruction to send data to a device, the program must force the output by one of several methods:

- Include the new line character (@) in the output. Do not use this method when coding MODE=LINE or XLATE=NO parameters.
- Code the XLATE=NO parameter.
- Specify SKIP=1 on a PRINTEXT instruction.
- Issue a TERMCTRL DISPLAY instruction.

Specifying Translation of Data

If you want to send or receive hexadecimal data (when issuing the PRINTEXT or READTEXT instructions) you must suppress translation of the data. The Series/1 uses EBCDIC, the GPIB Adaptor uses ASCII, and the GPIB devices use the code specific to each device type. To suppress translation of data, code the XLATE=NO parameter for PRINTEXT or READTEXT. If you want to send or receive data in EBCDIC for Series/1, ASCII for GPIB, or the code appropriate to a device, code XLATE=YES. Use of XLATE=NO with PRINTEXT forces output of data.

Specifying a User Buffer

If an application is to transfer more data than will fit in the system buffer (defined at system generation), the IOCB through which the application is enqueued must specify a user buffer. A user buffer is necessary if you want to specify the exact number of characters for a read.

In the following example, the IOCB (GPIBIOCB) specifies a BUFFER operand; the buffer length is 500 bytes. If a buffer is used, you are responsible for setting the buffer length (GPIBLEN in this example) to the number of bytes to be written via PRINTEXT. The buffer length is set to the number of bytes actually read by a READTEXT instruction. The example below shows how the buffer can be initialized for a PRINTEXT of 100 bytes.

```
          ENQT      GPIBIOCB                    CONNECT TO GPIB
          ..
          ..
          ..
          DEQT                                  DISCONNECT FROM GPIB
          ..
          ..
          ..
GPIBIOCB  IOCB      GPIB,BUFFER=GPIBUFF         GPIB TERMINAL NAME
GPIBBUFF  BUFFER    500,BYTES,INDEX=GPIBLEN     BUFFER FOR GPIB USE
          ..
          ..
          ..
          MOVE      GPIBLEN,100
```

If a user buffer is not desired, do not include the BUFFER parameter in the IOCB statement.

Initializing and Configuring the Bus

The TERMCTRL instruction provides operations that are used to initialize the bus: adapter reset (RSET), interface clear (IFC), and remote enable (REN). Normally these three operations are performed when a GPIB program starts, and need not be executed again unless an error occurs. When used at the start of a GPIB program, these control operations should be executed in the order discussed.

The RSET operation resets the adapter.

```
          TERMCTRL GPIB,RSET        RESET ADAPTER
```

The IFC operation clears the GPIB interface, thus causing the entire bus to be made inactive.

```
          TERMCTRL GPIB,IFC         INTERFACE CLEAR
```

The remote enable (REN) operation enables all the devices on the bus. Not all devices need to be enabled in this manner; some are initialized to an enabled state. Check the manufacturer's device description to see whether a remote enable is needed.

```
         TERMCTRL GPIB,REN          REMOTE ENABLE
         PRINTEXT LISTADDR          SEND LISTENER ADDRESS
         TERMCTRL DISPLAY           FORCE OUTPUT
         ..
         ..
         ..
LISTADDR TEXT     '%'               LISTEN ADDRESS FOR ONE DEVICE
```

Configuring the Bus

The configure bus (CON) operation of the TERMCTRL instruction does two things: it assigns the roles of talker and listener, and it can be used to write data to listeners.

In the following examples, the universal unlisten command (a question mark) is transmitted first, followed by a talker address, and then by the list of listener addresses. The unlisten command is a special message that resets all devices that were listeners to a nonlistening state.

```
Series/1 talks, and some device(s) listens:

         TERMCTRL GPIB,CON       CONFIGURE BUS
         PRINTEXT '?'            UNIVERSAL UNLISTEN COMMAND
         PRINTEXT 'U'            SERIES/1 TALK ADDRESS
         PRINTEXT LISTADDR       DEVICE LISTENER ADDRESS(ES)
         PRINTEXT '":'           DATA BLOCK TERMINATOR
         TERMCTRL DISPLAY        FORCE OUTPUT
         ..
         ..
         ..
         LISTADDR TEXT '%'       LISTEN ADDRESS

Series/1 sends data via CON:

         TERMCTRL GPIB,CON       CONFIGURE BUS
         PRINTEXT '?U'           (AS ABOVE)
         PRINTEXT LISTADDR       DEVICE LISTENER ADDRESS(ES)
         PRINTEXT '",'           DATA SEPARATOR
         PRINTEXT DATA           SEND DEVICE DATA
         PRINTEXT '":'           DATA BLOCK TERMINATOR
         TERMCTRL DISPLAY        FORCE OUTPUT
         ..
         ..
         ..
         LISTADDR TEXT '%'       LISTEN ADDRESS
```

```
Some device talks, and Series/1 listens:

         TERMCTRL GPIB,CON       CONFIGURE BUS
         PRINTEXT '?'            UNIVERSAL UNLISTEN COMMAND
         PRINTEXT TALKADDR       DEVICE TALK ADDRESS
         PRINTEXT '5'            SERIES/1 LISTENER ADDRESS
         PRINTEXT '":'           DATA BLOCK TERMINATOR
         TERMCTRL DISPLAY        FORCE OUTPUT
         ..
         ..
         ..
         TALKADDR TEXT 'E'       TALK ADDRESS

One device talks, and other device(s) listens:

         TERMCTRL GPIB,CON       CONFIGURE BUS
         PRINTEXT '?'            UNIVERSAL UNLISTEN COMMAND
         PRINTEXT TALKADDR       DEVICE TALK ADDRESS
         PRINTEXT LISTADDR       OTHER LISTENER ADDRESS(ES)
         PRINTEXT '":'           DATA BLOCK TERMINATOR
         TERMCTRL DISPLAY        FORCE OUTPUT
         ..
         ..
         ..
         LISTADDR TEXT '%'       LISTEN ADDRESS
         TALKADDR TEXT 'E'       TALK ADDRESS
```

Enqueuing and Dequeuing GPIB

The ENQT instruction connects Series/1 to a GPIB adapter and excludes other sources from accessing the adapter at the same time. You must specify the GPIB adapter that the Series/1 wants to communicate with. To do this, refer to the label of the IOCB instruction that identifies the GPIB adapter.

The DEQT instruction releases the GPIB adapter previously enqueued.

If your program is connected to a GPIB adapter and issues a a LOAD instruction, the task issuing the LOAD will be disconnected from the GPIB adapter, and the task being loaded will be connected to the GPIB adapter. If this is not desirable, the program issuing the LOAD must first issue a DEQT, followed by the LOAD, and then issue an ENQT to reconnect to the GPIB adapter.

In the following example, the IOCB (GPIBIOCB) is initially enqueued. Next, the program issues a DEQT to disconnect itself from the GPIB adapter. The program then issues a LOAD instruction for the program NEWPROG, which now has the GPIB adapter connected to it. After NEWPROG completes, the program reconnects to the GPIB adapter by issuing another ENQT.

```
          ENQT     GPIBIOCB            CONNECT TO GPIB
          ..
          ..
          ..
          DEQT                         DISCONNECT FROM GPIB
          LOAD     NEWPROG             LOAD NEW PROGRAM
          ENQT     GPIBIOCB            RECONNECT TO GPIB
          ..
          ..
          ..
GPIBIOCB  IOCB     GPIB
```

Series/1 Sending Data to a Device

The Series/1 acts as talker and transmits data to the listener(s) with a PRINTEXT instruction. The data actually goes to the GPIB adapter which in turn directs it to the designated device(s).

The following example shows what to code for Series/1 to send data to a device.

```
Example of writing character data:

          TERMCTRL GPIB,WRIT           WRITE TO GPIB
          PRINTEXT DATA                SEND DATA
          TERMCTRL DISPLAY             FORCE OUTPUT
          ..
          ..
          ..
DATA      TEXT     'IN;'               DEVICE DEPENDENT DATA
```

```
Example of writing hex data:
(note the simulated TEXT statement)

          TERMCTRL GPIB,WRIT           WRITE TO GPIB
          PRINTEXT HEXDATA,XLATE=NO    SEND HEX DATA
          TERMCTRL DISPLAY             FORCE OUTPUT
          ..
          ..
          ..
          DATA     X'0101'             TEXT LENGTH/COUNT
HEXDATA   DATA     X'03'               DEVICE DEPENDENT DATA
```

Series/1 Receiving Data from a Device

Series/1 receives data transfers from GPIB devices with a READTEXT instruction.

This example shows how the Series/1 receives data being sent by a device. It is assumed that the device appends an end-of-string character to the data to be sent. Since the device can send

a variable number of characters, the SE option is used in the READTEXT instruction to suppress incorrect length record exceptions.

```
TERMCTRL GPIB,READ,(SE,EOS),X'0D'     READ GPIB
READTEXT DATA                         READ INPUT DATA
```

Sending Data Between Devices

One device can send data to one or more other devices while the Series/1 merely monitors the data transfer. This is done with the READTEXT instruction. The following example shows how Series/1 allows the data transfer between the devices.

```
TERMCTRL GPIB,MON,(SE,EOS),X'0D'     READ MONITOR
READTEXT DATA                        DUMMY READ
```

Polling the Devices

Your program can perform either a serial or parallel polling operation to determine the source of a device interrupt. The TERMCTRL instruction performs polling.

Serial Polling Code these TERMCTRL instructions to perform serial polling (polling devices in sequence):

- TERMCTRL SPE (enable serial poll)
- TERMCTRL SPL (read serial poll)
- TERMCTRL SPD (disable serial poll).

The SPE command polls each device whose talker address has been specified. The SPL command reads the poll status of each device. The poll status returned for each device consists of two bytes: the talker address of the polled device (first byte) and the device status (second byte). The status byte is device-dependent, and is usually expressed in hex. If it is, use XLATE=NO on the appropriate READTEXT. The SPD command disables the serial poll status reporting ability of the devices previously enabled through an SPE command.

```
TERMCTRL GPIB,SPE           SERIAL POLL ENABLE
PRINTEXT TALKADDR           TALK ADDRESS OF DEVICE
TERMCTRL DISPLAY            FORCE OUTPUT
TERMCTRL GPIB,SPL           READ SERIAL POLL
READTEXT STATUS,XLATE=NO    GET SERIAL POLL STATUS
TERMCTRL GPIB,SPD           SERIAL POLL DISABLE
```

Parallel Polling: A parallel poll polls all enabled devices in parallel, with the poll status encoded as a single byte. Device response to parallel polling is highly device-dependent; many devices do not even acknowledge parallel polling.

Code the following TERMCTRL instructions to perform parallel polling:

- TERMCTRL PPE (enable parallel poll)
- TERMCTRL WPPL (write parallel poll command)
- TERMCTRL RPPL (read parallel poll status)
- TERMCTRL PPD (disable parallel poll)
- TERMCTRL PPU (unconfigure parallel poll).

The PPE command enables the devices to respond to a parallel poll. You must supply two bytes of information for each device to be polled. The first byte specifies the listener address. The first four bits of the second byte have, by definition, the value of six. The fifth bit is the line level (one or zero) which is the active level for the addressed device. The final three bits indicate which GPIB data line is used to request service. Bit values zero through seven are used to specify GPIB data lines one through eight.

Note: Some devices that respond to parallel polling may respond to a poll only in predefined ways. Consult your device manual.

Since a parallel poll word is most conveniently expressed in numeric form, PRINTEXT with XLATE=NO should be used. A TEXT statement must be simulated so that hex data can be used. In the following example, the TEXT statement is simulated by the first DATA statement containing the value X'0202'. This value corresponds to the length and count bytes of a TEXT statement followed by another DATA statement that contains the two bytes of polling information. In the example, a single device at listener address X'25' is enabled to respond at line level one on GPIB data line one.

```
          TERMCTRL GPIB,PPE              PARALLEL POLL ENABLE
          PRINTEXT PPEMSG,XLATE=NO       SEND DATA
          ..
          ..
          ..
          DATA     X'0202'               TEXT PARAMETERS
PPEMSG    DATA     X'2568'
```

The actual polling of the devices is accomplished by a write parallel poll command (WPPL). The resulting poll status byte is stored in the adapter until a read parallel poll status (RPPL) is performed. The RPPL operation moves the status byte into the byte specified on the TERMCTRL GPIB,RPPL statement.

```
         TERMCTRL GPIB,WPPL              WRITE PARALLEL POLL STATUS
         TERMCTRL GPIB,RPPL,,RPPLDATA    READ PARALLEL POLL STATUS
         ..
         ..
         ..
RPPLDATA DATA     X'00'                  PARALLEL POLL STATUS BYTE
```

One device or all devices configured to respond to parallel polling can be unconfigured using the parallel poll disable (PPD) or parallel poll unconfigure (PPU) command.

```
         TERMCTRL GPIB,PPD          PARALLEL POLL DISABLE
         PRINTEXT LISTADDR          LISTENER ADDRESS(ES)
         TERMCTRL DISPLAY           FORCE OUTPUT

                          or

         TERMCTRL GPIB,PPU          PARALLEL POLL UNCONFIGURE
```

Assigning Device Mode of Operation

Local Mode Operation: Some GPIB devices can be operated by an attached panel as well as via GPIB bus commands; this is called local mode operation. A device with this capability can be enabled to operate in local mode by the go to local mode TERMCTRL GTL command. Similarly, local mode can be disabled by the local lock out TERMCTRL LLO command. Both of these commands must specify the listener addresses of the devices to be operated in this mode.

```
         TERMCTRL GPIB,GTL          GO TO LOCAL
         PRINTEXT LISTADDR          LISTENER ADDRESS(ES)
         TERMCTRL DISPLAY           FORCE OUTPUT
         ..
         ..
         ..
         TERMCTRL GPIB,LLO          LOCAL LOCK OUT
         PRINTEXT LISTADDR          LISTENER ADDRESS(ES)
         TERMCTRL DISPLAY           FORCE OUTPUT
```

Device Group Operation: Some devices can have predefined operations which are executed by a trigger command. These operations can be initiated by a group execute trigger TERMCTRL GET command which must specify a list of listener addresses for the devices.

```
          TERMCTRL GPIB,GET          GROUP EXECUTE TRIGGER
          PRINTEXT LISTADDR          LISTENER ADDRESS(ES)
          TERMCTRL DISPLAY           FORCE OUTPUT
```

Coding GPIB Functions

These coding examples demonstrate the steps necessary to communicate with a device connected to the GPIB bus. In this sample case, the GPIB device is a plotter. Therefore, some operations unique to the plotter are covered.

Following this section is a sample program that contains the coding segments shown below. In the coding segments, the following plotter commands are used:

IN	Initialize plotter
PA x,y	Move pen to absolute position at x,y coordinates
OC	Output current pen position
IM	Initialize masks

Connecting to the GPIB Adapter

The program must first connect to the GPIB adapter with an ENQT instruction.

```
          ENQT      GPIBIOCB         CONNECT TO GPIB
          ..
          ..
          ..
GPIBIOCB  IOCB      GPIB             IOCB FOR GPIB ADAPTER
```

Initializing the Adapter

The program initializes the adapter and bus by means of the RSET and IFC operations.

```
          TERMCTRL  GPIB,RSET        RESET ADAPTER
          TERMCTRL  GPIB,IFC         CLEAR BUS
```

Enabling the GPIB Device

The plotter is remotely enabled with a REN command followed by its listener address.

```
          TERMCTRL  GPIB,REN         REMOTE ENABLE
          PRINTEXT  '%@'               THE PLOTTER
```

Programming for the GPIB Application *(continued)*

Clearing All Devices

All devices are cleared by a device clear command.

```
          TERMCTRL  GPIB,DCL          CLEAR ALL DEVICES
```

Preparing to Send Data

At this point, the bus is configured so that the Series/1 can send data to the plotter, that is, the Series/1 is enabled to talk, and the plotter is enabled to listen. The coding for this consists of the universal unlisten command (question mark), followed by the talker and listener addresses, the data block terminator, and a new line character (@) to force output.

```
          TERMCTRL  GPIB,CON          CONFIGURE BUS
          PRINTEXT  SENDCON           S/1 TALKS, PLOTTER LISTENS
          ..
          ..
          ..
SENDCON   TEXT      '?U%":@'
```

Writing Data to the Device

The Series/1 writes data to the plotter by issuing a GPIB write command. Then a series of plotter commands are sent. PRINTNUM is used to convert numeric data to character form.

```
          TERMCTRL  GPIB,WRIT,TO      WRITE, TIMER OVERRIDE
          PRINTEXT  'IN;'             INITIALIZE PLOTTER
          PRINTEXT  'PA'              MOVE PEN ABSOLUTE
          PRINTNUM  F5000             X=5000
          PRINTEXT  ','               SEPARATOR
          PRINTNUM  F5000             Y=5000
          PRINTEXT  ';'               TERMINATOR
          PRINTEXT  'OC;'             OUTPUT CURRENT PEN POSITION
          TERMCTRL  DISPLAY           FORCE OUTPUT
          ..
          ..
          ..
F5000     DATA      F'5000'
CURRPOS   TEXT      LENGTH=14
```

Preparing to Receive Data

Since the Series/1 requested data from the plotter, it must reconfigure the bus so that the plotter talks and the Series/1 listens.

```
          TERMCTRL  GPIB,CON          CONFIGURE BUS
          PRINTEXT  RECVCON           PLOTTER TALKS, S/1 LISTENS
          ..
          ..
          ..
RECVCON   TEXT      '?E5":@'
```

Receiving Data From the GPIB Device

The actual read is performed by a GPIB READ operation. Since a variable number of characters can be sent, incorrect length record (ILR) exceptions are suppressed with the SE option. The end of data is indicated by an end-of-string (EOS) character. At this point, the input data can be tested or displayed.

```
          TERMCTRL  GPIB,READ,(SE,EOS,TO),X'0D'
          READTEXT  CURRPOS           READ CURRENT POSITION
```

Preparing Device to Issue Service Request (SRQ) Interrupts

To show the use of service request (SRQ) interrupts, the plotter will issue an SRQ upon receipt of an invalid plotter command. First, the bus must be configured again.

```
          ENQT      GPIBIOCB          CONNECT TO GPIB
          TERMCTRL  GPIB,CON          CONFIGURE THE BUS
          PRINTEXT  SENDCON           S/1 TALKS, PLOTTER LISTENS
```

Then plotter commands can be transmitted which initialize the plotter to issue the SRQ when an error occurs, and to cause an error.

```
          TERMCTRL  GPIB,WRIT,TO      WRITE, TIMER OVERRIDE
          PRINTEXT  'IN;'             INITIALIZE PLOTTER
          PRINTEXT  'IM223,32,0;'     TO SEND SRQ UPON ERROR
          PRINTEXT  'XX;'             SEND INVALID COMMAND
          PRINTEXT  SKIP=1            FORCE OUTPUT
          DEQT                        DISCONNECT FROM GPIB
```

Receiving Device's SRQ Interrupt

At this point the plotter issues an SRQ interrupt. The following code receives the interrupt and posts the event. The preceding DEQT was necessary, since the attention list task would wait until the GPIB adapter was dequeued. Note that the ECB is initialized to the "event not occurred" condition. (This code appears elsewhere in the actual program sequence.)

```
          ATTNLIST   ($PF255,POSTSRQ),SCOPE=GLOBAL
          ..
          ..
          ..
POSTSRQ   EQU        *
          POST       SRQECB          POST EVENT
          ENDATTN
          ..
          ..
          ..
          SPACE
SRQECB    ECB        0
          SPACE
```

The program waits for the SRQ event to occur, and then continues operation.

```
          WAIT       SRQECB          WAIT FOR SRQ
          RESET      SRQECB          RESET ECB
```

Retrieving Device Status

To retrieve the plotter status, the program serially polls the plotter with the serial poll enable (SPE) and read serial poll (SPL) commands. The talker address for the plotter must be specified with the SPE. The SPL is accompanied by a READTEXT with XLATE=NO, because the plotter status is most easily handled in numeric form. Finally, a serial poll disable is issued to prevent the plotter from talking with status information.

```
          ENQT       GPIBIOCB          CONNECT TO GPIB
          TERMCTRL   GPIB,SPE          SERIAL POLL ENABLE
          PRINTEXT   'E@'              PLOTTER TALKER ADDRESS
          TERMCTRL   GPIB,SPL          READ SERIAL POLL
          READTEXT   STATUS,XLATE=NO   READ STATUS
          TERMCTRL   GPIB,SPD          SERIAL POLL DISABLE
          ..
          ..
          ..
STATUS    TEXT       LENGTH=2
```

GPIB Sample Program

Figure 74 shows the preceding coding segments as a complete sample program. An error testing routine is called after each I/O operation.

```
GPIBEG   PROGRAM  START
*
START    EQU      *
*
         ENQT     GPIBIOCB          CONNECT TO GPIB
         TERMCTRL GPIB,RSET         RESET ADAPTER
         CALL     ERRTEST           CHECK FOR ERRORS
         TERMCTRL GPIB,IFC          CLEAR BUS
         CALL     ERRTEST           CHECK FOR ERRORS
         TERMCTRL GPIB,REN          REMOTE ENABLE
         PRINTEXT '%@'                THE PLOTTER
         CALL     ERRTEST           CHECK FOR ERRORS
         TERMCTRL GPIB,DCL          CLEAR ALL DEVICES
         CALL     ERRTEST           CHECK FOR ERRORS
*
* CONFIGURE BUS SO THAT S/1 TALKS AND THE PLOTTER LISTENS.
*
         TERMCTRL GPIB,CON          CONFIGURE BUS
         PRINTEXT SENDCON           S/1 TALKS, PLOTTER LISTENS
         CALL     ERRTEST           CHECK FOR ERRORS
*
* INITIALIZE THE PLOTTER.
*
         TERMCTRL GPIB,WRIT,TO WRITE, TIMER OVERRIDE
         PRINTEXT 'IN;'             INITIALIZE PLOTTER
*
* MOVE THE PEN TO (5000,5000).
*
         PRINTEXT 'PA'              MOVE PEN ABSOLUTE
         PRINTNUM F5000             X=5000
         PRINTEXT ','               SEPARATOR
         PRINTNUM F5000             Y=5000
         PRINTEXT ';'               TERMINATOR
```

Figure 74 (Part 1 of 4). GPIB Sample Program

```
*
* COMMAND THE PLOTTER TO SEND ITS CURRENT POSITION.
*
         PRINTEXT  'OC;'            OUTPUT CURRENT POSITION
         TERMCTRL  DISPLAY          FORCE OUTPUT
         CALL      ERRTEST          CHECK FOR ERRORS
*
* CONFIGURE THE BUS SO THE PLOTTER TALKS, AND S/1 LISTENS.
*
         TERMCTRL  GPIB,CON         CONFIGURE BUS
         PRINTEXT  RECVCON          PLOTTER TALKS, S/1 LISTENS
         CALL      ERRTEST          CHECK FOR ERRORS
*
* READ THE CURRENT POSITION OF THE PLOTTER.
* (SPECIFY SUPPRESS EXCEPTION, TIMER OVERRIDE,
* END OF STRING CHAR = X'0D'
*
         TERMCTRL  GPIB,READ,(SE,EOS,TO),X'0D'
         READTEXT  CURRPOS          READ CURRENT POSITION
         CALL      ERRTEST          CHECK FOR ERRORS
*
* DISPLAY CURRENT POSITION OF PLOTTER.
*
         DEQT                       DISCONNECT FROM GPIB
         PRINTEXT  '@CURRENT PLOTTER POSITION = '
         PRINTEXT  CURRPOS          DISPLAY CURRENT POSITION
         PRINTEXT  SKIP=1
         EJECT
         ENQT      GPIBIOCB         CONNECT TO GPIB
*
* CONFIGURE THE BUS SO THAT S/1 TALKS AND PLOTTER LISTENS.
*
         TERMCTRL  GPIB,CON         CONFIGURE THE BUS
         PRINTEXT  SENDCON          S/1 TALKS, PLOTTER LISTENS
         CALL      ERRTEST          CHECK FOR ERRORS
```

Figure 74 (Part 2 of 4). GPIB Sample Program

```
*
* INITIALIZE THE PLOTTER TO SEND AN SRQ UPON AN ERROR.
*
         TERMCTRL  GPIB,WRIT,TO      WRITE, TIMER OVERRIDE
         PRINTEXT  'IN;'             INITIALIZE PLOTTER
         PRINTEXT  'IM223,32,0;'     TO SEND SRQ UPON ERROR
*
* CAUSE A PLOTTER ERROR.
*
         PRINTEXT  'XX;'             SEND INVALID COMMAND
         PRINTEXT  SKIP=1            FORCE OUTPUT
         CALL      ERRTEST           CHECK FOR ERRORS
*
         DEQT                        DISCONNECT FROM GPIB
*
* WAIT FOR THE SRQ, AND POLL THE PLOTTER TO GET ITS STATUS.
*
         WAIT      SRQECB            WAIT FOR SRQ
         RESET     SRQECB            RESET EVENT
         ENQT      GPIBIOCB          CONNECT TO GPIB
         TERMCTRL  GPIB,SPE          SERIAL POLL ENABLE
         PRINTEXT  'E@'              PLOTTER TALKER ADDRESS
         CALL      ERRTEST           CHECK FOR ERRORS
         TERMCTRL  GPIB,SPL          READ SERIAL POLL
         READTEXT  STATUS,XLATE=NO   READ STATUS
         CALL      ERRTEST           CHECK FOR ERRORS
         TERMCTRL  GPIB,SPD          SERIAL POLL DISABLE
         CALL      ERRTEST           CHECK FOR ERRORS
*
         DEQT      GPIBIOCB          DISCONNECT FROM GPIB
*
* DISPLAY PLOTTER STATUS.
*
         PRINTEXT  '@PLOTTER STATUS = '
         PRINTNUM  STATUS,MODE=HEX   DISPLAY STATUS
         PRINTEXT  SKIP=1
*
         PROGSTOP
```

Figure 74 (Part 3 of 4). GPIB Sample Program

```
*
* THIS ATTENTION LIST TASK RECEIVES CONTROL WHEN
* AN SRQ INTERRUPT IS POSTED.
*
         ATTNLIST  ($PF255,POSTSRQ),SCOPE=GLOBAL
*
POSTSRQ  EQU       *
         POST      SRQECB              POST EVENT
         ENDATTN
*
* SUBROUTINE TO CHECK FOR AND DISPLAY GPIB TERMINAL ERRORS.
*
         SUBROUT   ERRTEST
*
         MOVE      TASKRC,GPIBEG       GET TASK RETURN CODE
         IF        (TASKRC,NE,-1)      ERROR?
           DEQT                        DISCONNECT FROM GPIB
           PRINTEXT '@GPIB TERMINAL I/O ERROR ='
           PRINTNUM TASKRC,MODE=HEX
           PRINTEXT SKIP=1
           PROGSTOP
         ENDIF
*
         RETURN
*
TASKRC   DATA      F'0'
GPIBIOCB IOCB      GPIB                IOCB FOR GPIB ADAPTER
SRQECB   ECB       0                   SRQ EVENT
F5000    DATA      F'5000'
CURRPOS  TEXT      LENGTH=14
STATUS   TEXT      LENGTH=2
*
* CONFIGURATION INFORMATION.
*
SENDCON  TEXT      '?U%":@'            S/1 TALKS, PLOTTER LISTENS
RECVCON  TEXT      '?E5":@'            PLOTTER TALKS, S/1 LISTENS
*
         ENDPROG
         END
```

Figure 74 (Part 4 of 4). GPIB Sample Program

General Purpose Interface Bus - IEEE Standard 488-1975

Interacting with the GPIB Application (Using $GPIBUT1)

The $GPIBUT1 utility enables you to interactively control and transfer data to and from GPIB devices. This utility can also be used as a diagnostic tool to check out the application program interface and the attached devices.

The $GPIBUT1 commands are listed below. To obtain this list at your terminal, enter a question mark in response to the prompting message, COMMAND(?):

```
COMMAND(?):  ?

CH   - DEFINE END CHARACTER
CP   - CHANGE GPIB PARTITION
DD   - DEFINE DEVICE
EN   - END THE PROGRAM
GP   - GPIB CONTROL
LDCB - LIST DEVICE CONTROL BLOCK
RE   - READ DATA
RS   - RESET I/O ADAPTER
ST   - READ ERROR STATUS
SU   - SUSPEND PROGRAM
WR   - WRITE DATA
¶¶ATTN - PGPIB◀◀ TO POST
¶¶ATTN - GPRESUME◀◀ TO RESUME PROGRAM

COMMAND(?):
```

If a $GPIBUT1 command fails, use the attention list command PGPIB to terminate the failing operation. If the PGPIB command is used, you must issue an RS command (or RSET if GP subcommands are used) to reset the adapter.

Defining End Character (CH)

The CH command defines or changes the ending character that is added to output data.

```
COMMAND(?):  CH

CHARACTER TO BE APPENDED TO OUTPUT DATA -- NOW IS NONE

1 = CARRIAGE RETURN
2 = LINE FEED
3 = END OF TEXT
4 = USER SPECIFIED HEX BYTE
5 = NONE

SELECT CODE:  3

  END CHARACTER IS NOW ETX

COMMAND(?):
```

Interacting with the GPIB Application (Using $GPIBUT1) *(continued)*

Changing GPIB Partition (CP)

The CP command changes the partition to which the GPIB adapter is connected. The partition is initially defined at system generation.

```
COMMAND(?):  CP 2
  PARTITION CHANGED TO 2
COMMAND(?):  CP
  PARTITION NUMBER (NOW IS 2):
  PARTITION NUMBER NOT CHANGED
COMMAND(?):
```

Defining Device (DD)

The DD command prompts for the name of the GPIB adapter. This name is specified in the TERMINAL configuration statement. The name specified is used for all enqueues of the adapter until another DD command is issued.

```
COMMAND(?):  DD
NEW GPIB TERMINAL NAME =  GPIB
COMMAND(?):
```

Ending The Program (EN)

The EN command ends the $GPIBUT1 utility.

```
COMMAND(?):  EN
```

Controlling GPIB (GP)

The GP command enables you to enter the GPIB bus command options that can be specified on the TERMCTRL instruction, as described in the *Language Reference*.

When GP is entered and followed by a bus command, $GPIBUT1 prompts for additional data, depending upon the specific command. For example, the CON (configure) command requires both configuration and programming data. For the REN (remote enable) command, a list of GPIB device addresses must be included.

Where appropriate, $GPIBUT1 performs PRINTEXT/READTEXT operations as part of the execution of a GP command, inserting delimiters as needed. In some cases, one delimiter is a user-defined end character. The end character can be defined by the CH command.

```
COMMAND(?):  GP

GPIB COMMAND(?):  CON

OPTION(SE,EOS,TO,EOI):  TO
OPTION(SE,EOS,TO,EOI):

CONFIGURATION DATA:  ?U%
PROGRAMMING DATA (OR NONE):  IN;
PROGRAMMING DATA (OR NONE):  OE;
PROGRAMMING DATA (OR NONE):

CONFIGURATION DATA:  ?E5
PROGRAMMING DATA (OR NONE):  NONE
PROGRAMMING DATA (OR NONE):
```

```
CONFIGURATION DATA:

GPIB COMMAND(?):  READ
OPTION(SE,EOS,TO,EOI):  SE
                         WARNING - EOS OR EOI REQUIRED ...
OPTION(SE,EOS,TO,EOI):  EOS
                         WARNING - SE MAY BE NEEDED ...
  EOS BYTE (HEX):  OD             (X'OD')
OPTION(SE,EOS,TO,EOI):
TRANSLATE INPUT?  Y

HOW MANY CHARACTERS (MAX=DEFAULT=80):
VALUE DEFAULTED TO 80

0

COMMAND(?):
```

Listing Device Control Block (LDCB)

The LDCB command lists the contents of the current GPIB device control block (DCB). The DCB describes the last GPIB operation performed. However, the information provided may require that you use the GPIB Adapter manual. The items listed include:

- Address of the GPIB terminal control block (CCB)
- Address of the GPIB device control block (DCB)
- Status of the DCB control word, specifically:
  - Cycle steal status key (that is, the address space of the data buffer)
  - GPIB operation mnemonic (for an undefined operation, '****')

  - Status of the chaining, input, suppress exception (SE), end of string (EOS), timer override (TO), and end of identify (EOI) bits, if they are set.

- End of string character
- Address of the residual status block (RSB)
- Chain address
- Byte count for the data transfer
- Address of the data buffer
- Contents of the data buffer, expressed as:
  - A string of hexadecimal words
  - EBCDIC characters
  - ASCII characters.

The DCB is checked for certain error conditions, including:

- DCB words two or three not equal to zero
- RSB address not equal to zero, and suppress exception set
- Chain address nonzero and chaining bit set.

If the byte count is odd, the last byte in the string of hex words is not part of the buffer and should be disregarded. Because the buffer data can be either EBCDIC or ASCII, depending on the application, it is displayed in both character codes. In most cases, the ASCII data displayed will be accurate. An inappropriate translation is displayed as a blank line.

The following example illustrates a DCB used in the execution of:

```
TERMCTRL GPIB,CON,TO
PRINTEXT '?U%":@'
```

```
COMMAND (?):  LDCB

DISPLAY OF DCB FOR GPIB TERMINAL GPIB

GPIB CCB AT ADDRESS 1058
GPIB DCB AT ADDRESS OFFE

CONTROL WORD
  CYCLE STEAL KEY IS 0
  TIMER OVERRIDE SET
  DEVICE OPERATION IS CON

BYTE COUNT IS   5
DATA ADDRESS IS 1102

DATA IN HEX FORMAT IS:

3F55 2522 2C00

DATA INTERPRETED AS EBCDIC IS:

?U%'':

COMMAND (?):
```

Reading Data (RE)

The RE command reads data from the GPIB adapter. You can also specify GPIB options (TERMCTRL functions), translation, and the number of characters to be read.

```
COMMAND(?):  READ
OPTION(SE,EOS,TO,EOI):  SE
                        WARNING - EOS OR EOI REQUIRED ...
OPTION(SE,EOS,TO,EOI):  EOS
                        WARNING - SE MAY BE NEEDED ...
  EOS BYTE (HEX):  0D           (X'0D')
OPTION(SE,EOS,TO,EOI):
TRANSLATE INPUT?  Y

HOW MANY CHARACTERS (MAX=DEFAULT=80):
VALUE DEFAULTED TO 80

0

COMMAND(?):
```

RS - Resetting the GPIB Adapter (RS)

The RS command issues a device reset to the adapter. Any pending interrupt or busy condition is cleared when this command is executed.

```
COMMAND(?):  RS

  RESET ADAPTER?  Y

COMMAND(?):
```

Reading Error Status (ST)

The ST command displays the status information contained in the adapter cycle steal status words and the residual status block (RSB).

```
COMMAND(?):  ST

READ STATUS?  Y

CYCLE STEAL STATUS BLOCK (HEX)

  RESIDUAL ADDRESS =                  B45A
  RESIDUAL BYTE COUNT =               0050
  (RESERVED) =                        0000
  (RESERVED) =                        0000
  ERROR STATUS =                      8000
  BUS STATUS (AFTER POWER ON) =       0008
  BUS STATUS (CURRENT) =              000A
  SPE DEVICE ADDRESS =                0000
  DCB SPECIFICATION CHECK =           0000
  (RESERVED) =                        0000
  DCB ADDRESS =                       1238

RESIDUAL STATUS BLOCK (HEX)

  RESIDUAL BYTE COUNT = 0000
  RSB FLAGS =           0000
  (RESERVED) =          0000
  (RESERVED) =          0000
  (RESERVED) =          0000

RESET ADAPTER?  Y

NO DATA RECEIVED

COMMAND(?):
```

Suspending $GPIBUT1 (SU)

The SU command suspends the operation of $GPIBUT1 until you tell it to resume using GPRESUME. This enables you to run $GPIBUT1 concurrently with a GPIB application from the same terminal.

```
COMMAND(?):  SU

$GPIBUT1 SUSPENDED
```

Writing Data (WR)

The WR command writes data to the GPIB adapter. You can specify GPIB options (TERMCTRL functions) and translation.

```
COMMAND(?):  WR
OPTION(SE,EOS,TO,EOI):
  WRITE HEX DATA?  N
  ENTER TEXT:  IN;
  IS THE DATA OK?  Y
COMMAND(?):  WR
OPTION(SE,EOS,TO,EOI):
  WRITE HEX DATA?  Y
  ENTER HEX WORDS:  0A0D 0102
  IS THE DATA OK?  Y
COMMAND(?):
```

Posting GPIB Operation Completion (PGPIB)

Sometimes a GPIB operation waits indefinitely, such as if a device fails to respond to an operation in which timer override (TO) is specified. The PGPIB attention command can be used to complete the operation; it cancels the operation by simulating its completion. However, after a PGPIB, the GPIB adapter is still in a busy state, so it must be reset.

```
GPIB COMMAND (?):  READ
OPTION (SE,EOS,TO,EOI):  TO
TRANSLATE INPUT?  Y
HOW MANY CHARACTERS (MAX=DEFAULT=80):
VALUE DEFAULTED TO 80
> PGPIB
DATA RECEIVED:
GPIB COMMAND (?):  RSET
  RESET ADAPTER?  Y
```

Resuming $GPIBUT1 Operation (GPRESUME)

If $GPIBUT1 has been suspended using the SU command, the GPRESUME attention command can be used to resume it.

```
> GPRESUME

$GPIBUT1 RESUMED

COMMAND (?):
```

Debugging Applications with $GPIBUT1

Along with tools such as $DEBUG, $GPIBUT1 can be used to debug an executing GPIB application. This can be done from one terminal by using the suspend and resume commands.

Using $L, load both $GPIBUT1 and the application program from a terminal. (The application program can also be loaded using $DEBUG.) First load $GPIBUT1 and then suspend it with the SU command. Application debugging can then proceed until you need to use a $GPIBUT1 function. At that point, use GPRESUME to reactivate $GPIBUT1.

The list DCB (LDCB) and display status (ST) commands are helpful in this context because they interpret the effects of GPIB operations. In addition, PGPIB can be used to complete a GPIB operation which is in an indefinite wait state.

$GPIBUT1 Utility Example

Figure 75 shows many of the $GPIBUT1 utility operations. The attached device is a plotter.

```
> $CP 2
> $L $GPIBUT1
$GPIBUT1     45P, LP= 0000
$GPIBUT1 - GPIB UTILITY
USING GPIB TERMINAL GPIB1
COMMAND(?):  DD
NEW GPIB TERMINAL NAME =  GPIB3
  GPIB3 IS NOT A GPIB TERMINAL
RETRY?  Y
```

Figure 75 (Part 1 of 9). GPIB Utility Example

```
NEW GPIB TERMINAL NAME =  GPIB
COMMAND (?):  CP 2
  PARTITION CHANGED TO 2
COMMAND (?):  CH
CHARACTER TO BE APPENDED TO OUTPUT DATA -- NOW IS NONE
 1 = CARRIAGE RETURN
 2 = LINE FEED
 3 = END OF TEXT
 4 = USER SPECIFIED HEX BYTE
 5 = NONE
SELECT CODE:  3
  END CHARACTER NOW IS ETX
```

Figure 75 (Part 2 of 9). GPIB Utility Example

```
COMMAND (?):  CH

CHARACTER TO BE APPENDED TO OUTPUT DATA -- NOW IS ETX

 1 = CARRIAGE RETURN
 2 = LINE FEED
 3 = END OF TEXT
 4 = USER SPECIFIED HEX BYTE
 5 = NONE

SELECT CODE:  5

  END CHARACTER NOW IS NONE
COMMAND (?):  GP

GPIB COMMAND (?):  CON

OPTION(SE,EOS,TO,EOI):  TO
OPTION(SE,EOS,TO,EOI):

CONFIGURATION DATA:  ?U%
PROGRAMMING DATA (OR NONE):  IN;
PROGRAMMING DATA (OR NONE):  OE;
PROGRAMMING DATA (OR NONE):

CONFIGURATION DATA:  ?E5
PROGRAMMING DATA (OR NONE):  NONE
PROGRAMMING DATA (OR NONE):
CONFIGURATION DATA:
```

Figure 75 (Part 3 of 9). GPIB Utility Example

```
GPIB COMMAND (?):  READ
OPTION(SE,EOS,TO,EOI):  SE
                         WARNING - EOS OR EOI REQUIRED ...
OPTION(SE,EOS,TO,EOI):  EOS
                         WARNING - SE MAY BE NEEDED ...
  EOS BYTE (HEX):  0D(X'0D)
OPTION(SE,EOS,TO,EOI):
TRANSLATE INPUT?  Y

HOW MANY CHARACTERS (MAX=DEFAULT=80):
VALUE DEFAULTED TO 80

0

GPIB COMMAND(?):  CON

OPTION(SE,EOS,TO,EOI):

CONFIGURATION DATA:  ?U%
PROGRAMMING DATA (OR NONE):  IN;
PROGRAMMING DATA (OR NONE):  IM223,32,0;
PROGRAMMING DATA (OR NONE):  XX;

******  SRQ RECEIVED  ******
```

Figure 75 (Part 4 of 9). GPIB Utility Example

```
PROGRAMMING DATA (OR NONE):

CONFIGURATION DATA:
GPIB COMMAND (?):  SPE

OPTION(SE,EOS,TO,EOI):

  TALKER ADDRESS LIST:  E

  IS THE DATA OK?  Y

GPIB COMMAND (?):  SPL

WARNING - SPE MUST HAVE JUST BEEN EXECUTED

TRANSLATE INPUT?  N

DATA RECEIVED:
 4578

WARNING - AN SPD MAY NOW BE REQUIRED
```

Figure 75 (Part 5 of 9). GPIB Utility Example

```
GPIB COMMAND (?):  SPD

OPTION(SE,EOS,TO,EOI):

GPIB COMMAND (?):  READ

OPTION(SE,EOS,TO,EOI):  TO

TRANSLATE INPUT?  Y

HOW MANY CHARACTERS (MAX=DEFAULT=80):

VALUE DEFAULTED TO 80
```

Figure 75 (Part 6 of 9). GPIB Utility Example

```
> PGPIB

DATA RECEIVED:


GPIB COMMAND (?):  READ

OPTION(SE,EOS,TO,EOI):
TRANSLATE INPUT?  Y

HOW MANY CHARACTERS (MAX=DEFAULT=80)
VALUE DEFAULTED TO 80

ERROR CODE = 0002

GPIB BUSY
READ STATUS?  N

  RESET ADAPTER?  Y
```

Figure 75 (Part 7 of 9). GPIB Utility Example

```
NO DATA RECEIVED
GPIB COMMAND (?):  READ

OPTION(SE,EOS,TO,EOI):
TRANSLATE INPUT?  Y

HOW MANY CHARACTERS (MAX=DEFAULT=80)

VALUE DEFAULTED TO 80

ERROR CODE = 0180

  EXCEPTION ON INPUT
  DEVICE DEPENDENT STATUS AVAILABLE

READ STATUS?  Y
```

Figure 75 (Part 8 of 9). GPIB Utility Example

```
CYCLE STEAL STATUS BLOCK (HEX)

  RESIDUAL ADDRESS =                B45A
  RESIDUAL BYTE COUNT =             0050
  (RESERVED) =                      0000
  (RESERVED) =                      0000
  ERROR STATUS =                    8000
  BUS STATUS (AFTER POWER ON) =     0008
  BUS STATUS (CURRENT) =            000A
  SPE DEVICE ADDRESS =              0000
  DCB SPECIFICATION CHECK =         0000
  (RESERVED) =                      0000
  DCB ADDRESS =                     1238
RESIDUAL STATUS BLOCK (HEX)

  RESIDUAL BYTE COUNT = 0000
  RSB FLAGS =           0000
  (RESERVED) =          0000
  (RESERVED) =          0000
  (RESERVED) =          0000

  RESET ADAPTER?  Y

NO DATA RECEIVED

GPIB COMMAND (?):  END
```

Figure 75 (Part 9 of 9). GPIB Utility Example

Detecting Errors During GPIB Operations

To control the GPIB bus and the attached devices, you should check the return code after each operation. In general, the application program performs error recovery by retrieving and analyzing the adapter cycle steal status block or residual status block. The manual *General Purpose Interface Bus (GPIB) Adapter - RPQ D02118 Custom Feature* contains detailed information on cycle steal status and the residual status block. The methods available to the application program for detecting possible errors and retrieving the return codes returned are described below.

GPIB errors can be detected in the same manner as other EDX terminal errors: by testing the first word of the task control block after an I/O instruction, or by coding an error routine (identified by TERMERR= in the PROGRAM statement). Except for return code 3 (busy after reset), the application program should handle GPIB return codes like other terminal errors.

An application program can initialize a GPIB bus by means of the TERMCTRL GPIB,RSET statement. This generates an interface clear (IFC) bus command. All GPIB devices on the bus must initialize themselves in response to this command. If the application program issues another command before a device can complete initialization, the busy after reset condition will occur. One solution is to cause the program to wait long enough for the reset operation to complete (this takes about 350 milliseconds). Another is to code the TERMCTRL GPIB command (which follows the reset) with the timer override (TO) option.

For exception conditions, the first byte of the error code indicates whether a read or write operation was requested. A value of 1 in the first byte indicates a read exception, while a 2 indicates a write exception. The second byte of the error code is the interrupt status byte (ISB) which contains further information on the exception.

Examining Interrupt Status Byte

The ISB can be examined to determine whether the exception condition resulted from an application program, hardware, or system error. Unless otherwise noted, the ISB bits below describe the condition when the bit is on.

Bit 0 — Unless bit 2 is on, indicates that cycle steal status should be retrieved via the TERMCTRL GPIB,STAT statement.

Bit 1 — Indicates a delayed command rejection which suggests a system error or an inappropriate/ unusual TERMCTRL GPIB statement.

Bit 2 — Indicates an incorrect length record, which means that the number of characters read from GPIB was less than specified in the input buffer. If the system buffer (contained in the terminal control block) was used, then the associated TERMCTRL GPIB,READ statement should contain the suppress exceptions (SE) option.

Bit 3 — Indicates a device control block specification error, which may be a system error. Word 8 of the cycle steal status block indicates the cause.

Bit 4 Indicates a storage data check, which means that there is a parity problem with main storage.

Bit 5 Indicates an invalid storage address was passed to the GPIB adapter. This can be a system problem or a problem with the application program buffer.

Bit 6 Indicates that the GPIB adapter tried to use an invalid storage key. This can be a system problem or a problem with the application program buffer.

Bit 7 Indicates that an interface data parity error was detected.

Examining Cycle Steal Status Block

If bit zero of the ISB is on, word 4 of the cycle steal status block can be examined to determine the source of the error. The bits in word 4 given below describe the condition when the bit is on.

Bit 0 The GPIB adapter timed out while waiting to receive data from the bus. To prevent this, specify timer override (TO) on the TERMCTRL GPIB,READ statement.

Bit 2 The GPIB adapter timed out while waiting to send data to the bus. To prevent this, specify timer override (TO) on the TERMCTRL GPIB,WRIT statement.

Bit 4 End of string (EOS) was specified on a TERMCTRL GPIB,READ statement, but the buffer was filled before the EOS character was received.

Bit 5 End of information (EOI) was specified on a TERMCTRL GPIB,READ statement, but the buffer was filled before EOI was received.

Retrieving Cycle Steal Status

When a GPIB operation terminates with an exception, an 11-word cycle steal status block is available. To retrieve it, the application should issue a TERMCTRL GPIB,STAT instruction specifying an area in storage to contain the status block.

```
          TERMCTRL GPIB,STAT,,STATDATA    RETRIEVE STATUS
           *
           *
           *
STATDATA DATA     11F'0'                  BLOCK FOR STATUS
```

Retrieving Residual Status Block

If a GPIB READ or read monitor (MON) operation specifies the suppress exception (SE) option, the 5-word residual status block is available. To retrieve it the application should issue a TERMCTRL GPIB,RSB instruction specifying an area in storage to contain the status block.

```
          TERMCTRL GPIB,RSB,,RSBDATA    RETRIEVE STATUS
           *
           *
           *
RSBDATA   DATA     5F'0'                BLOCK FOR STATUS
```

Notes

Glossary of Terms and Abbreviations

This glossary defines terms and abbreviations used in the Series/1 Event Driven Executive software publications. All software and hardware terms pertain to EDX. This glossary also serves as a supplement to the *IBM Data Processing Glossary*, GC20-1699.

$SYSLOGA, $SYSLOGB. The name of the alternate system logging device. This device is optional but, if defined, should be a terminal with keyboard capability, not just a printer.

$SYSLOG. The name of the system logging device or operator station; must be defined for every system. It should be a terminal with keyboard capability, not just a printer.

$SYSPRTR. The name of the system printer.

abend. Abnormal end-of-task. Termination of a task prior to its completion because of an error condition that cannot be resolved by recovery facilities while the task is executing.

ACCA. See asynchronous communications control adapter.

address key. Identifies a set of Series/1 segmentation registers and represents an address space. It is one less than the partition number.

address space. The logical storage identified by an address key. An address space is the storage for a partition.

application program manager. The component of the Multiple Terminal Manager that provides the program management facilities required to process user requests. It controls the contents of a program area and the execution of programs within the area.

application program stub. A collection of subroutines that are appended to a program by the linkage editor to provide the link from the application program to the Multiple Terminal Manager facilities.

asynchronous communications control adapter. An ASCII terminal attached via #1610, #2091 with #2092, or #2095 with #2096 adapters.

attention key. The key on the display terminal keyboard that, if pressed, tells the operating system that you are entering a command.

attention list. A series of pairs of 1 to 8 byte EBCDIC strings and addresses pointing to EDL instructions. When the attention key is pressed on the terminal, the operator can enter one of the strings to cause the associated EDL instructions to be executed.

backup. A copy of data to be used in the event the original data is lost or damaged.

base record slots. Space in an indexed file that is reserved for based records to be placed.

base records. Records are placed into an indexed file while in load mode or inserted in process mode with a new high key.

basic exchange format. A standard format for exchanging data on diskettes between systems or devices.

binary synchronous device data block (BSCDDB). A control block that provides the information to control one Series/1 Binary Synchronous Adapter. It determines the line characteristics and provides dedicated storage for that line.

Glossary of Terms and Abbreviations

block. (1) See data block or index block. (2) In the Indexed Method, the unit of space used by the access method to contain indexes and data.

block mode. The transmission mode in which the 3101 Display Station transmits a data data stream, which has been edited and stored, when the SEND key is pressed.

BSCAM. See binary synchronous communications access method.

binary synchronous communications access method. A form of binary synchronous I/O control used by the Series/1 to perform data communications between local or remote stations.

BSCDDB. See binary synchronous device data block.

buffer. An area of storage that is temporarily reserved for use in performing an input/output operation, into which data is read or from which data is written. See input buffer and output buffer.

bypass label processing. Access of a tape without any label processing support.

CCB. See terminal control block.

central buffer. The buffer used by the Indexed Access Method for all transfers of information between main storage and indexed files.

character image. An alphabetic, numeric, or special character defined for an IBM 4978 Display Station. Each character image is defined by a dot matrix that is coded into eight bytes.

character image table. An area containing the 256 character images that can be defined for an IBM 4978 Display Station. Each character image is coded into eight bytes, the entire table of codes requiring 2048 bytes of storage.

character mode. The transmission mode in which the 3101 Display Station immediately sends a character when a keyboard key is pressed.

cluster. In an indexed file, a group of data blocks that is pointed to from the same primary-level index block, and includes the primary-level index block. The data records and blocks contained in a cluster are logically contiguous, but are not necessarily physically contiguous.

COD (change of direction). A character used with ACCA terminal to indicate a reverse in the direction of data movement.

cold start. Starting the spool facility by erasing any spooled jobs remaining in the spool data set from any previous spool session.

command. A character string from a source external to the system that represents a request for action by the system.

common area. A user-defined data area that is mapped into the partitions specified on the SYSTEM definition statement. It can be used to contain control blocks or data that will be accessed by more than one program.

completion code. An indicator that reflects the status of the execution of a program. The completion code is displayed or printed on the program's output device.

constant. A value or address that remains unchanged thoughout program execution.

controller. A device that has the capability of configuring the GPIB bus by designating which devices are active, which devices are listeners, and which device is the talker. In Series/1 GPIB implementation, the Series/1 is always the controller.

conversion. See update.

control station. In BSCAM communications, the station that supervises a multipoint connection, and performs polling and selection of its tributary stations. The status of control station is assigned to a BSC line during system generation.

cross-partition service. A function that accesses data in two partitions.

cross-partition supervisor. A supervisor in which one or more supervisor modules reside outside of partition 1 (address space 0).

data block. In an indexed file, an area that contains control information and data records. These blocks are a multiple of 256 bytes.

data record. In an indexed file, the records containing customer data.

data set. A group of records within a volume pointed to by a directory member entry in the directory for the volume.

data set control block (DSCB). A control block that provides the information required to access a data set, volume or directory using READ and WRITE.

data set shut down. An indexed data set that has been marked (in main storage only) as unusable due to an error.

DCE. See directory control entry.

device data block (DDB). A control block that describes a disk or diskette volume.

direct access. (1) The access method used to READ or WRITE records on a disk or diskette device by specifying their location relative the beginning of the data set or volume. (2) In the Indexed Access Method, locating any record via its key without respect to the previous operation. (3) A condition in terminal I/O where a READTEXT or a PRINTEXT is directed to a buffer which was previously enqueued upon by an IOCB.

directory. (1) A series of contiguous records in a volume that describe the contents in terms of allocated data sets and free space. (2) A series of contiguous records on a device that describe the contents in terms of allocated volumes and free space. (3) For the Indexed Access Method Version 2, a data set that defines the relationship between primary and secondary indexed files (secondary index support).

directory control entry (DCE). The first 32 bytes of the first record of a directory in which a description of the directory is stored.

directory member entry (DME). A 32-byte directory entry describing an allocated data set or volume.

display station. An IBM 4978, 4979, or 3101 display terminal or similar terminal with a keyboard and a video display.

DME. See directory member entry.

DSCB. See data set control block.

dynamic storage. An increment of storage that is appended to a program when it is loaded.

end-of-data indicator. A code that signals that the last record of a data set has been read or written. End-of-data is determined by an end-of-data pointer in the DME or by the physical end of the data set.

ECB. See event control block.

EDL. See Event Driven Language.

emulator. The portion of the Event Driven Executive supervisor that interprets EDL instructions and performs the function specified by each EDL statement.

end-of-tape (EOT). A reflective marker placed near the end of a tape and sensed during output. The marker signals that the tape is nearly full.

enter key. The key on the display terminal keyboard that, if pressed, tells the operating system to read the information you entered.

event control block (ECB). A control block used to record the status (occurred or not occurred) of an event; often used to synchronize the execution of tasks. ECBs are used in conjunction with the WAIT and POST instructions.

Event Driven Language (EDL). The language for input to the Event Driven Executive compiler ($EDXASM), or the Macro and Host assemblers in conjunction with the Event Driven Executive macro libraries. The output is interpreted by the Event Driven Executive emulator.

EXIO (execute input or output). An EDL facility that provides user controlled access to Series/1 input/output devices.

external label. A label attached to the outside of a tape that identifies the tape visually. It usually contains items of identification such as file name and number, creation data, number of volumes, department number, and so on.

external name (EXTRN). The 1- to 8-character symbolic EBCDIC name for an entry point or data field that is not defined within the module that references the name.

FCA. See file control area.

FCB. See file control block.

file. A set of related records treated as a logical unit. Although file is often used interchangeably with data set, it usually refers to an indexed or a sequential data set.

file control area (FCA). A Multiple Terminal Manager data area that describes a file access request.

file control block (FCB). The first block of an indexed file. It contains descriptive information about the data contained in the file.

file control block extension. The second block of an indexed file. It contains the file definition parameters used to define the file.

file manager. A collection of subroutines contained within the program manager of the Multiple Terminal Manager that provides common support for all disk data transfer operations as needed for transaction-oriented application programs. It supports indexed and direct files under the control of a single callable function.

floating point. A positive or negative number that can have a decimal point.

formatted screen image. A collection of display elements or display groups (such as operator prompts and field input names and areas) that are presented together at one time on a display device.

free pool. In an indexed data set, a group of blocks that can be used for either data blocks or index blocks. These differ from other free blocks in that these are not initially assigned to specific logical positions in the file.

free space. In an indexed file, records blocks that do not currently contain data, and are available for use.

free space entry (FSE). An 8-byte directory entry defining an area of free space within a volume or a device.

FSE. See free space entry.

general purpose interface bus. The IEEE Standard 488-1975 that allows various interconnected devices to be attached to the GPIB adapter (RPQ D02118).

Glossary of Terms and Abbreviations

GPIB. See general purpose interface bus.

group. A unit of 100 records in the spool data set allocated to a spool job.

H exchange format. A standard format for exchanging data on diskettes between systems or devices.

host assembler. The assembler licensed program that executes in a 370 (host) system and produces object output for the Series/1. The source input to the host assembler is coded in Event Driven Language or Series/1 assembler language. The host assembler refers to the System/370 Program Preparation Facility (5798-NNQ).

host system. Any system whose resources are used to perform services such as program preparation for a Series/1. It can be connected to a Series/1 by a communications link.

IACB. See indexed access control block.

IAR. See instruction address register.

ICB. See indexed access control block.

IIB. See interrupt information byte.

image store. The area in a 4978 that contains the character image table.

immediate data. A self-defining term used as the operand of an instruction. It consists of numbers, messages or values which are processed directly by the computer and which do not serve as addresses or pointers to other data in storage.

index. In an indexed file, an ordered collection of pairs of keys and pointers, used to sequence and locate records.

index block. In an indexed file, an area that contains control information and index entries. These blocks are a multiple of 256 bytes.

indexed access control block (IACB/ICB). The control block that relates an application program to an indexed file.

indexed access method. An access method for direct or sequential processing of fixed-length records by use of a record's key.

indexed data set. Synonym for indexed file.

indexed file. A file specifically created, formatted and used by the Indexed Access Method. An indexed file is sometimes called an indexed data set.

index entry. In an indexed file, a key-pointer pair, where the pointer is used to locate a lower-level index block or a data block.

index register (#1, #2). Two words defined in EDL and contained in the task control block for each task. They are used to contain data or for address computation.

input buffer. (1) See buffer. (2) In the Multiple Terminal Manager, an area for terminal input and output.

input output control block (IOCB). A control block containing information about a terminal such as the symbolic name, size and shape of screen, the size of the forms in a printer, or an optional reference to a user provided buffer.

instruction address register (IAR). The pointer that identifies the machine instruction currently being executed. The Series/1 maintains a hardware IAR to determine the Series/1 assembler instruction being executed. It is located in the level status block (LSB).

integer. A positive or negative number that has no decimal point.

interactive. The mode in which a program conducts a continuous dialogue between the user and the system.

internal label. An area on tape used to record identifying information (similar to the identifying information placed on an external label). Internal labels are checked by the system to ensure that the correct volume is mounted.

interrupt information byte (IIB). In the Multiple Terminal Manager, a word containing the status of a previous input/output request to or from a terminal.

invoke. To load and activate a program, utility, procedure, or subroutine into storage so it can run.

job. A collection of related program execution requests presented in the form of job control statements, identified to the jobstream processor by a JOB statement.

job control statement. A statement in a job that specifies requests for program execution, program parameters, data set definitions, sequence of execution, and, in general, describes the environment required to execute the program.

job stream processor. The job processing facility that reads job control statements and processes the requests made by these statements. The Event Driven Executive job stream processor is $JOBUTIL.

jumper. (1) A wire or pair of wires which are used for the arbitrary connection between two circuits or pins in an attachment card. (2) To connect wire(s) to an attachment card or to connect two circuits.

key. In the Indexed Access Method, one or more consecutive characters used to identify a record and establish its order with respect to other records. See also key field.

key field. A field, located in the same position in each record of an indexed file, whose content is used for the key of a record.

level status block (LSB). A Series/1 hardware data area that contains processor status. This area is eleven words in length.

library. A set of contiguous records within a volume. It contains a directory, data sets and/or available space.

line. A string of characters accepted by the system as a single input from a terminal; for example, all characters entered before the carriage return on the teletypewriter or the ENTER key on the display station is pressed.

link edit. The process of resolving external symbols in one or more object modules. A link edit is performed with $EDXLINK whose output is a loadable program.

listener. A controller or active device on a GPIB bus that is configured to accept information from the bus.

load mode. In the Indexed Access Method, the mode in which records are loaded into base record slots in an indexed file.

load module. A single module having cross references resolved and prepared for loading into storage for execution. The module is the output of the $UPDATE or $UPDATEH utility.

load point. (1) Address in the partition where a program is loaded. (2) A reflective marker placed near the beginning of a tape to indicate where the first record is written.

lock. In the Indexed Access Method, a method of indicating that a record or block is in use and is not available for another request.

logical screen. A screen defined by margin settings, such as the TOPM, BOTM, LEFTM and RIGHTM parameters of the TERMINAL or IOCB statement.

LSB. See level status block.

mapped storage. The processor storage that you defined on the SYSTEM statement during system generation.

member. A term used to identify a named portion of a partitioned data set (PDS). Sometimes member is also used as a synonym for a data set. See data set.

menu. A formatted screen image containing a list of options. The user selects an option to invoke a program.

menu-driven. The mode of processing in which input consists of the responses to prompting from an option menu.

message. In data communications, the data sent from one station to another in a single transmission. Stations communication with a series of exchanged messages.

multifile volume. A unit of recording media, such as tape reel or disk pack, that contains more than one data file.

multiple terminal manager. An Event Driven Executive licensed program that provides support for transaction-oriented applications on a Series/1. It provides the capability to define transactions and manage the programs that support those transactions. It also manages multiple terminals as needed to support these transactions.

multivolume file. A data file that, due to its size, requires more than one unit of recording media (such as tape reel or disk pack) to contain the entire file.

new high key. A key higher than any other key in an indexed file.

nonlabeled tapes. Tapes that do not contain identifying labels (as in standard labeled tapes) and contain only files separated by tapemarks.

null character. A user-defined character used to define the unprotected fields of a formatted screen.

option selection menu. A full screen display used by the Session Manager to point to other menus or system functions, one of which is to be selected by the operator. (See primary option menu and secondary option menu.)

output buffer. (1) See buffer. (2) In the Multiple Terminal Manager, an area used for screen output and to pass data to subsequent transaction programs.

overlay. The technique of reusing a single storage area allocated to a program during execution. The storage area can be reused by loading it with overlay programs that have been specified in the PROGRAM statement of the program or by calling overlay segments that have been specified in the OVERLAY statement of $EDXLINK.

overlay area. A storage area within a program reserved for overlay programs specified in the PROGRAM statement or overlay segments specified in the OVERLAY statement in $EDXLINK.

overlay program. A program in which certain control sections can use the same storage location at different times during execution. An overlay program can execute concurrently as an asynchronous task with other programs and is specified in the EDL PROGRAM statement in the main program.

overlay segment. A self-contained portion of a program that is called and sequentially executes as a synchronous task. The entire program that calls the overlay segment need not be maintained in storage while the overlay segment is executing. An overlay segment is specified in the OVERLAY statement of $EDXLINK or $XPSLINK (for initialization modules).

overlay segment area. A storage area within a program or supervisor reserved for overlay segments. An overlay segment area is specified with the OVLAREA statement of $EDXLINK.

Glossary of Terms and Abbreviations

parameter selection menu. A full screen display used by the Session Manager to indicate the parameters to be passed to a program.

partition. A contiguous fixed-sized area of storage. Each partition is a separate address space.

performance volume. A volume whose name is specified on the DISK definition statement so that its address is found during IPL, increasing system performance when a program accesses the volume.

physical timer. Synonym for timer (hardware).

polling. In data communications, the process by which a multipoint control station asks a tributary if it can receive messages.

precision. The number of words in storage needed to contain a value in an operation.

prefind. To locate the data sets or overlay programs to be used by a program and to store the necessary information so that the time required to load the prefound items is reduced.

primary file. An indexed file containing the data records and primary index.

primary file entry. For the Indexed Access Method Version 2, an entry in the directory describing a primary file.

primary index. The index portion of a primary file. This is used to access data records when the primary key is specified.

primary key. In an indexed file, the key used to uniquely identify a data record.

primary-level index block. In an indexed file, the lowest level index block. It contains the relative block numbers (RBNs) and high keys of several data blocks. See cluster.

primary menu. The program selection screen displayed by the Multiple Terminal Manager.

primary option menu. The first full screen display provided by the Session Manager.

primary station. In a Series/1 to Series/1 attachment, the processor that control communication between the two computers. Contrast with secondary station.

primary task. The first task executed by the supervisor when a program is loaded into storage. It is identified by the PROGRAM statement.

priority. A combination of hardware interrupt level priority and a software ranking within a level. Both primary and secondary tasks will execute asynchronously within the system according to the priority assigned to them.

process mode. In the Indexed Access Method, the mode in which records can be retrieved, updated, inserted or deleted.

processor status word (PSW). A 16-bit register used to (1) record error or exception conditions that may prevent further processing and (2) hold certain flags that aid in error recovery.

program. A disk- or diskette-resident collection of one or more tasks defined by a PROGRAM statement; the unit that is loaded into storage. (See primary task and secondary task.)

program header. The control block found at the beginning of a program that identifies the primary task, data sets, storage requirements and other resources required by a program.

program/storage manager. A component of the Multiple Terminal Manager that controls the execution and flow of application programs within a single program area and contains the support needed to allow multiple operations and sharing of the program area.

protected field. A field in which the operator cannot use the keyboard to enter, modify, or erase data.

PSW. See processor status word.

QCB. See queue control block.

QD. See queue descriptor.

QE. See queue element.

queue control block (QCB). A data area used to serialize access to resources that cannot be shared. See serially reusable resource.

queue descriptor (QD). A control block describing a queue built by the DEFINEQ instruction.

queue element (QE). An entry in the queue defined by the queue descriptor.

quiesce. To bring a device or a system to a halt by rejection of new requests for work.

quiesce protocol. A method of communication in one direction at a time. When sending node wants to receive, it releases the other node from its quiesced state.

record. (1) The smallest unit of direct access storage that can be accessed by an application program on a disk or diskette using READ and WRITE. Records are 256 bytes in length. (2) In the Indexed Access Method, the logical unit that is transferred between $IAM and the user's buffer. The length of the buffer is defined by the user. (3) In BSCAM communications, the portions of data transmitted in a message. Record length (and, therefore, message length) can be variable.

recovery. The use of backup data to re-create data that has been lost or damaged.

reflective marker. A small adhesive marker attached to the reverse (nonrecording) surface of a reel of magnetic tape. Normally, two reflective markers are used on each reel of tape. One indicates the beginning of the recording area on the tape (load point), and the other indicates the proximity to the end of the recording area (EOT) on the reel.

relative block address (RBA). The location of a block of data on a 4967 disk relative to the start of the device.

relative record number. An integer value identifying the position of a record in a data set relative to the beginning of the data set. The first record of a data set is record one, the second is record two, the third is record three.

relocation dictionary (RLD). The part of an object module or load module that is used to identify address and name constants that must be adjusted by the relocating loader.

remote management utility control block (RCB). A control block that provides information for the execution of remote management utility functions.

reorganize. The process of copying the data in an indexed file to another indexed file in a manner that rearranges the data for more optimum processing and free space distribution.

restart. Starting the spool facility w the spool data set contains jobs from a previous session. The jobs in the spool data set can be either deleted or printed when the spool facility is restarted.

return code. An indicator that reflects the results of the execution of an instruction or subroutine. The return code is usually placed in the task code word (at the beginning of the task control block).

roll screen. A display screen which is logically segmented into an optional history area and a work area. Output directed to the screen starts display at the beginning of the work area and continues on down in a line-by-line sequence. When the work area gets full, the operator presses ENTER/SEND and its contents are shifted into the optional history area and the work area itself is erased. Output now starts again at the beginning of the work area.

SBIOCB. See sensor based I/O control block.

second-level index block. In an indexed data set, the second-lowest level index block. It contains the addresses and high keys of several primary-level index blocks.

secondary file. See secondary index.

secondary index. For the Indexed Access Method Version 2, an indexed file used to access data records by their secondary keys. Sometimes called a secondary file.

secondary index entry. For the Indexed Access Method Version 2, this an an entry in the directory describing a secondary index.

secondary key. For the Indexed Access Method Version 2, the key used to uniquely identify a data record.

secondary option menu. In the Session Manager, the second in a series of predefined procedures grouped together in a hierarchical structure of menus. Secondary option menus provide a breakdown of the functions available under the session manager as specified on the primary option menu.

secondary task. Any task other than the primary task. A secondary task must be attached by a primary task or another secondary task.

secondary station. In a Series/1 to Series/1 attachment, the processor that is under the control of the primary station.

sector. The smallest addressable unit of storage on a disk or diskette. A sector on a 4962 or 4963 disk is equivalent to an Event Driven Executive record. On a 4964 or 4966 diskette, two sectors are equivalent to an Event Driven Executive record.

selection. In data communications, the process by which the multipoint control station asks a tributary station if it is ready to send messages.

self-defining term. A decimal, integer, or character that the computer treats as a decimal, integer, or character and not as an address or pointer to data in storage.

sensor based I/O control block (SBIOCB). A control block containing information related to sensor I/O operations.

sequential access. The processing of a data set in order of occurrence of the records in the data set. (1) In the Indexed Access Method, the processing of records in ascending collating sequence order of the keys. (2) When using READ/WRITE, the processing of records in ascending relative record number sequence.

serially reusable resource (SRR). A resource that can only be accessed by one task at a time. Serially reusable resources are usually managed via (1) a QCB and ENQ/DEQ statements or (2) an ECB and WAIT/POST statements.

service request. A device generated signal used to inform the GPIB controller that service is required by the issuing device.

session manager. A series of predefined procedures grouped together as a hierarchical structure of menus from which you select the utility functions, program preparation facilities, and language processors needed to prepare and execute application programs. The menus consist of a primary option menu that displays functional groupings and secondary option menus that display a breakdown of these functional groupings.

shared resource. A resource that can be used by more than one task at the same time.

Glossary of Terms and Abbreviations

shut down. See data set shut down.

source module/program. A collection of instructions and statements that constitute the input to a compiler or assembler. Statements may be created or modified using one of the text editing facilities.

spool job. The set of print records generated by a program (including any overlays) while engueued to a printer designated as a spool device.

spool session. An invocation and termination of the spool facility.

spooling. The reading of input data streams and the writing of output data streams on storage devices, concurrently with job execution, in a format convenient for later processing or output operations.

SRQ. See service request.

stand-alone dump. An image of processor storage written to a diskette.

stand-alone dump diskette. A diskette supplied by IBM or created by the $DASDI utility.

standard labels. Fixed length 80-character records on tape containing specific fields of information (a volume label identifying the tape volume, a header label preceding the data records, and a trailer label following the data records).

static screen. A display screen formatted with predetermined protected and unprotected areas. Areas defined as operator prompts or input field names are protected to prevent accidental overlay by input data. Areas defined as input areas are not protected and are usually filled in by an operator. The entire screen is treated as a page of information.

station. In BSCAM communications, a BSC line attached to the Series/1 and functioning in a point-to-point or multipoint connection. Also, any other terminal or processor with which the Series/1 communicates.

subroutine. A sequence of instructions that may be accessed from one or more points in a program.

supervisor. The component of the Event Driven Executive capable of controlling execution of both system and application programs.

system configuration. The process of defining devices and features attached to the Series/1.

SYSGEN. See system generation.

system generation. The processing of defining I/O devices and selecting software options to create a supervisor tailored to the needs of a specific Series/1 hardware configuration and application.

system partition. The partition that contains the root segment of the supervisor (partition number 1, address space 0).

talker. A controller or active device on a GPIB bus that is configured to be the source of information (the sender) on the bus.

tape device data block (TDB). A resident supervisor control block which describes a tape volume.

tapemark. A control character recorded on tape used to separate files.

task. The basic executable unit of work for the supervisor. Each task is assigned its own priority and processor time is allocated according to this priority. Tasks run independently of each other and compete for the system resources. The first task of a program is the primary task. All tasks attached by the primary task are secondary tasks.

task code word. The first two words (32 bits) of a task's TCB; used by the emulator to pass information from system to task regarding the outcome of various operations, such as event completion or arithmetic operations.

task control block (TCB). A control block that contains information for a task. The information consists of pointers, save areas, work areas, and indicators required by the supervisor for controlling execution of a task.

task supervisor. The portion of the Event Driven Executive that manages the dispatching and switching of tasks.

TCB. See task control block.

terminal. A physical device defined to the EDX system using the TERMINAL configuration statement. EDX terminals include directly attached IBM displays, printers and devices that communicate with the Series/1 in an asynchronous manner.

terminal control block (CCB). A control block that defines the device characteristics, provides temporary storage, and contains links to other system control blocks for a particular terminal.

terminal environment block (TEB). A control block that contains information on a terminal's attributes and the program manager operating under the Multiple Terminal Manager. It is used for processing requests between the terminal servers and the program manager.

terminal screen manager. The component of the Multiple Terminal Manager that controls the presentation of screens and communications between terminals and transaction programs.

terminal server. A group of programs that perform all the input/output and interrupt handling functions for terminal devices under control of the Multiple Terminal Manager.

terminal support. The support provided by EDX to manage and control terminals. See terminal.

timer. The timer features available with the Series/1 processors. Specifically, the 7840 Timer Feature card (4955 only) or the native timer (4952, 4954, and 4956). Only one or the other is supported by the Event Driven Executive.

trace range. A specified number of instruction addresses within which the flow of execution can be traced.

transaction oriented applications. Program execution driven by operator actions, such as responses to prompts from the system. Specifically, applications executed under control of the Multiple Terminal Manager.

transaction program. See transaction-oriented applications.

transaction selection menu. A Multiple Terminal Manager display screen (menu) offering the user a choice of functions, such as reading from a data file, displaying data on a terminal, or waiting for a response. Based upon the choice of option, the application program performs the requested processing operation.

tributary station. In BSCAM communications, the stations under the supervision of a control station in a multipoint connection. They respond to the control station's polling and selection.

unmapped storage. The processor storage in your processor that you did not define on the SYSTEM statement during system generation.

unprotected field. A field in which the operator can use the keyboard to enter, modify or erase data. Also called non-protected field.

update. (1) To alter the contents of storage or a data set. (2) To convert object modules, produced as the output of an assembly or compilation, or the output of the linkage editor, into a form that can be loaded into storage for program execution and to update the directory of the volume on which the loadable program is stored.

user exit. (1) Assembly language instructions included as part of an EDL program and invoked via the USER instruction. (2) A point in an IBM-supplied program where a user written routine can be given control.

variable. An area in storage, referred to by a label, that can contain any value during program execution.

vary offline. (1) To change the status of a device from online to offline. When a device is offline, no data set can be accessed on that device. (2) To place a disk or diskette in a state where it is unknown by the system.

vary online. To place a device in a state where it is available for use by the system.

vector. An ordered set or string of numbers.

volume. A disk, diskette, or tape subdivision defined using $INITDSK or $TAPEUT1.

volume descriptor entry (VDE). A resident supervisor control block that describes a volume on a disk or diskette.

volume label. A label that uniquely identifies a single unit of storage media.

Index

The following index contains entries for this book only. See the *Library Guide and Common Index* for a Common Index to all Event Driven Executive books.

Index

C

Index

S

Index

IBM Series/1 Event Driven Executive

Publications Order Form

Instructions:

1. Complete the order form, supplying all of the requested information. (Please print or type.)
2. If you are placing the order by phone, dial **1-800-IBM-2468.**
3. If you are mailing your order, fold the order form as indicated, seal with tape, and mail. We pay the postage.

Ship to:

Name:

Address:

City:

State: Zip:

Bill to:

Customer number:

Name:

Address:

City:

State: Zip:

Your Purchase Order No.:

Phone: ( )

Signature:

Date:

Order:

Description	Order number	Qty.
Reference books:		
Set of the following six books. To order individual copies, use the following order numbers.	SBOF-1627	
Communications Guide	SC34-0638	
Extended Address Mode and Performance Analyzer User Guide	SC34-0591	
Installation and System Generation Guide	SC34-0646	
Language Reference	SC34-0643	
Library Guide and Common Index	SC34-0645	
Messages and Codes	SC34-0636	
Operator Commands and Utilities Reference	SC34-0644	
Guides and reference cards:		
Set of the following four books and reference cards. To order individual copies, use the following order numbers.	SBOF-1628	
Customization Guide	SC34-0635	
Event Driven Language Programming Guide	SC34-0637	
Operation Guide	SC34-0642	
Problem Determination Guide	SC34-0639	
Language Reference Card	SX34-0165	
Operator Commands and Utilities Reference Card	SX34-0164	
Conversion Charts Reference Card	SX34-0163	
Reference Card Envelope	SX34-0166	
Set of three reference cards and storage envelope. (One set is included with order number SBOF-1627)	SBOF-1629	
Binders:		
3-ring easel binder with 1 inch rings	SR30-0324	
3-ring easel binder with 2 inch rings	SR30-0327	
Standard 3-ring binder with 1 inch rings	SR30-0329	
Standard 3-ring binder with 1 1/2 inch rings	SR30-0330	
Standard 3-ring binder with 2 inch rings	SR30-0331	
Diskette binder (Holds eight 8-inch diskettes.)	SB30-0479	

Publications Order Form

Cut or Fold Along Line

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, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

IBM Corporation
1 Culver Road
Dayton, New Jersey 08810

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

International Business Machines Corporation

IBM Series/1 Event Driven Executive
Communications Guide

Order No. SC34-0638-0

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SC34-0638-0
Printed in U.S.A.

Cut or Fold Along Line

Reader's Comment Form

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, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Information Development, Department 28B
3405 (Internal Zip)
P.O. Box 1328
Boca Raton, Florida 33432

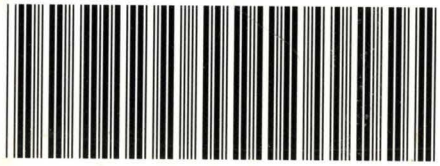
Fold and tape | Please Do Not Staple | Fold and tape

IBM®

IBM

International Business Machines Corporation

SC34-0638-0



SC34-0638-0
Program Numbers: 5719-XS5, 5719-XX6,
5719-LM6, 5719-CX1, 5799-PGH
File Number: S1-30
Printed in U.S.A.