

SC34-0316-2

File No. S1-30

LICENSED PROGRAM

IBM Series/1

Event Driven Executive

Communications and Terminal
Applications Guide

Program Numbers: 5719-LM5 5719-LM6 5719-MS1

5719-UT3 5719-UT4 5719-XS1 5719-XS2 5719-XX2 5719-XX3 5740-LM2 5740-LM3 SC34-0316-2

File No. S1-30

LICENSED PROGRAM

# IBM Series/1 Event Driven Executive

Communications and Terminal Applications Guide

Program Numbers: 5719-LM5 5719-LM6 5719-MS1

5719-UT3 5719-UT4 5719-XS1 5719-XS2 5719-XX2 5719-XX3 5740-LM2 5740-LM3

#### Third Edition (APRIL 1980)

Use this publication only for the purpose stated.

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems, refer to the latest <u>IBM Series/1 Graphic Bibliography</u>, GA34-0055, for the editions that are applicable and current.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services which 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 reader's comments is provided at the back of this publication. If the form has been removed, address your comments to IBM Corporation, Systems Publications, Department 27T, P.O. Box 1328, Boca Raton, Florida 33432. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

# SUMMARY OF AMENDMENTS

#### Terminal Support

Terminal support information for the 3101 Display Terminal (Models 1 and Model 2) was added to Chapter 1.

# Multiple Terminal Manager

Chapter 5 has been modified for PL/I and 3101 Display Terminal as provided by the Multiple Terminal Manager.

# Remote Management Utility (Version 2 only)

Chapter 6 is a new chapter that describes the Remote Management Utility.

## Bibliography

The Bibliography lists the books in the Event Driven Executive library and a recommended reading sequence. Other publications related to the Event Driven Executive are also listed.

# Miscellaneous Changes

This manual has been modified to include new function and to improve technical accuracy and clarity. Additional material and technical changes are indicated by vertical bars in the left margin.

#### HOW TO USE THIS BOOK

The material in this section is a guide to the use of this book. It defines the purpose, audience, and content of the book as well as listing aids for using the book and background materials.

#### **PURPOSE**

The <u>IBM Series/1</u> <u>Event Driven Executive Communications and Terminal Applications Guide</u>, SC34-0316 describes how to use the Event Driven Executive to communicate with interactive devices, such as, terminals or other processors.

This manual provides extensions to the <u>System Guide</u>, the <u>Utilities</u>, <u>Operator Commands</u>, <u>Program Preparation</u>, <u>Messages and Codes</u>, and the <u>Language Reference</u> manuals.

#### AUDIENCE

This book is written for system and application programmers with considerable knowledge in BSC and host operation, including IBM and non-IBM communications hardware.

To write applications for remotely attached devices, you must be familiar with line control procedures. Experience in programming realtime programs in the Event Driven Language is required. Experience coding programs in assembler language for the Series/1 will enable you to extend the terminal application capabilities of the system.

#### HOW THIS BOOK IS ORGANIZED

This book is organized into two parts. The first part explains criteria for selecting communications methods or techniques available with the Event Driven Executive system. The second part consists of individual chapters which describe how to design particular communication techniques.

The topics covered in part two include:

Terminal Support

Binary Synchronous Communications

Host Communications Facility

Multiple Terminal Manager

Remote Management Utility

Graphics

1

#### EXAMPLES AND OTHER AIDS

Throughout this book, coding examples and illustrations are used to clarify coding techniques and requirements. Coding examples are fully executable portions of complete programs that may be entered as they are shown. Coding illustrations are non-executable portions of incomplete programs that show the correct format of all required parameters on a statement. Missing code, or code provided by you, is indicated by a series of vertical or horizontal dots.

Several other aids are provided to assist you in using this book:

- A Summary of Amendments lists the significant changes made to this publication since the last edition
- A Bibliography:
  - Lists the books in the Event Driven Executive library along with a brief description of each book and a recommended reading sequence
  - Lists related publications and materials
- A Glossary which defines terms
- A Common Index which includes entries from each book in the Event Driven Executive library

References to other manuals are made throughout this manual using shortened titles. For the full title and order number of manuals mentioned in the text, see the Bibliography.

#### RELATED PUBLICATIONS

Related publications are listed in the Bibliography.

# SUBMITTING AN APAR

If you have a problem with the Series/1 Event Driven Executive services, you are encouraged to fill out an authorized program analysis report (APAR) form as described in the IBM Series/1 Authorized Program Analysis Report (APAR) User's Guide, GC34-0099.

# CONTENTS

	Introduction	•	•	1
	Part I - Technique Selection Criteria	•	•	3
	Chapter 1. Techniques Available Using the Event Driven			
	Executive			5
	Terminal Support			5
	Terminology for Supported Terminals			7
	Binary Synchronous Communications Access Method (BSCAM)			9
	Host Communications Facility			9
	Multiple Terminal Manager			9
l	Remote Management Utility			
	Graphics			
	Utilities	•	•	10
	Part II - Technique Design Information		•	11
	Chapter 2. Terminal Support			13
	Terminal Operations			
	Terminal I/O Instructions			
	Data Formatting Instructions			
	Terminal Definition Functions			
	Interrupt Processing Functions			
	Considerations for Feature #1610 or #2091 with #2092	•	•	Ι,
	Adapter			17
	Return Codes			
	Considerations for Feature #2095 with #2096 Adapter			
	#7850 Teletypewriter Adapter			
	Special Considerations for the IBM 3101 in Character	•	•	2 1
	Mode			22
	Special Considerations for the IBM 3101 in Block Mode			25
1	Interprocessor Communications			
	Hardware Preparation			
	Terminal Control Block (CCB)	•	•	
	Transmission Protocol	•	•	31
	Transmission Protocol	,•	•	2.7
	CRDELAY=			
	CUDIYPE=	•	•	3 3
	Chapter 3. Binary Synchronous Communications		-	35
	Access Level			
	Conversational Operations			36
	Multipoint Operations			36
	Task Control			37
	The Event Driven Language BSC Statements			38
	BSCCLOSE			38
	BSCIOCB			39
	BSCLINE	•	•	42
	BSCOPEN			44
	BSCREAD	•	•	
	BSCREAD Types	_	_	46

BSCWRITE	49
BSCWRITE Types	50
Error Recovery	56
Sample Program: Write Transparent	59
Sample Program: Read Transparent	60
Utility Programs (BSC)	61
•	
\$BSCTRCE	61
\$BSCUT1	62
\$BSCUT2	
RWI - Read/Write Non-transparent Data	66
RWIX - Read/Write Transparent Data	67
RWIXMP - Read/Write Transparent, Multidrop Line	67
RI - Read Transparent/Non-transparent	68
WI - Write Non-transparent	
WIX - Write Transparent	
	70
<del>v</del>	
CH - Change Hardcopy Device	70
RWIVX - Read/Write Transparent Conversational	70
RWIV - Read/Write Non-transparent Conversational	71
\$PRT2780 and \$PRT3780 Utility Programs	72
\$RJE2780 and \$RJE3780 Utility Programs	73
Chapter 4. Host Communications Facility	81
Open Series/1 Data Sets	82
Host Data Set Naming Conventions	82
Host Data Set Characteristics	83
Host System Considerations	83
Record Sizes	83
Variable Length Records	84
Data Transfer Rates	84
System Status Data Set	85
TP Statement	88
Examples of Use	88
TP Statement Syntax	90
	90
TP FETCH	92
TP OPENIN	93
TP OPENOUT	94
TP READ	95
TP RELEASE	96
TP SET	97
TP SUBMIT	98
	100
	101
	102
,	105
	106
	107
READDATA	801
READ80 and READOBJ	109
SET, FETCH, and RELEASE	110
	111
	112
Chapter 5. Multiple Terminal Manager	113

	Introduction	. 113
	Hardware Requirements	. 114
	Software Requirements	
	Program Operation Overview	
	Program Management	. 115
	Terminal/Screen Management	
	File Management	
	Multiple Terminal Manager Operation	
	Multiple Terminal Manager Initialization Program	
	Terminal Server Programs	
	Application Program Manager	
	Multiple Terminal Manager Utilities	
	Sign-On/Sign-Off	
	Data Files	
	Application Program Interface	
ŀ	Considerations for the IBM 3101 Model 2 Terminal	
ŀ	Multiple Terminal Manager Components	
	Program Execution	
	User Program Organization	
	Input Buffer Address	
	Output Buffer Address	
	Terminal Environment Block (TEB)	. 128
	Interrupt Information Byte (IIB)	
	Controlling the Logic Flow of Programs	
	CALL ACTION	
	CALL LINK	
	CALL LINKON	
	CALL CYCLE	
	Communicating with ASCII Terminals	
	CALL WRITE	
	Communicating with IBM 4978/4979/3101 Displays	
	CALL SETPAN	. 134
	CALL CHGPAN	. 135
	CALL SETCUR	. 137
	CALL BEEP	. 137
	CALL MENU	. 137
Ì	CALL FTAB	. 138
	CALL FAN	. 139
,	Accessing the Terminal Environment Block	
	CALL CDATA	
	Disk File Support	. 140
	CALL FILEIO	
	Event Driven Executive Direct File I/O Considerations	146
	FILEIO Indexed Access Method Considerations	
	Programming Considerations	
	Event Driven Language Programming Considerations	. 151
	FORTRAN Programming Considerations	. 152
	COBOL Programming Considerations	
ŀ	PL/I Programming Considerations	
1	SIGNON/SIGNOFF Programs	
	SIGNON	
	SIGNOFF	
	Operator Interface	
	Multiple Terminal Manager Initiation and Termination	158
	singing UD	1 m ×

Program Initiation and Termination	158
Utilities	159
Distribution, Installation and Program Preparation	161
Installation	162
Program Preparation	164
Event Driven Language Program Preparation	164
FORTRAN Program Preparation	165
COBOL Program Preparation	166
PL/I Program Preparation	167
Storage Requirements	168
System Generation Considerations	169
·	
Volume Requirements	169
Data Set Requirements	171
MTMSTORE	171
TERMINAL	171
Screen Format Volume - SCRNS	173
User Application Program Volume - PRGRMS	173
SIGNONFL	174
Multiple Terminal Manager Defaults and How to Change	177
Multiple Terminal Manager Messages	178
Example - File Maintenance Transaction Application	182
EDL Sample Prog1	190
EDL Sample Prog2	191
•	193
	195
FORTRAN Sample Progl	197
FORTRAN Sample Prog2	198
PL/I Sample Progl	200
PL/I Sample Prog2	202
Chapter 6. Remote Management Utility	205
Remote Management Functions	206
Hardware Requirements	207
Software Requirements	207
Remote Management Utility Interface	207
	208
Binary Synchronous Communication Protocol	
Record Exchange	208
Record Format	209
Record Blocking	211
Buffer Allocation	211
Parameter Passing	212
Remote Management Utility Functional Operation	213
ALLOCATE Function	214
Required Field Descriptions	214
DELETE Function	216
Required Field Descriptions	~
DUMP Function	216
	216 218
	218
Required Field Descriptions	218 218
Required Field Descriptions	218 218 220
Required Field Descriptions	218 218 220 220
Required Field Descriptions	218 218 220 220 223
Required Field Descriptions	218 218 220 220 223 223
Required Field Descriptions	218 218 220 220 223 223 225
Required Field Descriptions	

Passthru Record Types	232
Text or Program Function Key	232
Request for Data	236
Program End	236
No Data	236
PASSTHRU Blocking	237
Considerations on Using PASSTHRU	237
	243
RECEIVE Function	244
Required Field Descriptions	
SEND Function	247
Required Field Descriptions	248
SHUTDOWN Function	251
Required Field Descriptions	251
WRAP Function	254
Required Field Descriptions	254
Count Record	256
Data Record	257
Status Record	258
Sample Host Programs	259
Error Handling	277
Types of Errors	277
Error Messages	279
Installation	281
Remote Management Utility Modules	281
System Generation Requirements	281
Storage Requirements	282
Remote Management Utility Defaults	283
Modifying Defaults	283
Host ID	284
Remote ID	285
BSC Device Address	286
Communications Line	287
Storage	288
Buffer Size	289
Standard Data Set	290
Source Data Set	290
Passthru Data	291
CDRRM Equate Listing	292
Alankan 7 Anantiisa	~~~
Chapter 7. Graphics	299
General Description	299
Hardware Considerations	300
Appendix A. Code Types	303
Bibliography	309
Event Driven Executive Library Summary	309
Event Driven Executive Library	309
Summary of Library	310
System Guide	310
Utilities	310
Language Reference	311
Communications Guide	311
Internal Design	311
Peference Summary	312

Tabs			312
Reading Sequence			312
Other Event Driven Execu	tive Programming	Publications	313
Other Series/1 Programmi	ng Publications		313
Other Programming Public	ations		314
Series/1 System Library	Publications .	• • • • • • • • •	314
Glossary			317
Common Index			329

# LIST OF FIGURES

Figure	1.	Supported Devices and Features	6
Figure	2.	Terminal I/O - ACCA Return Codes	20
Figure	3.	Terminal I/O - Interprocessor Communications	
Returr	1 Cod	des	31
Figure	4.	Required Buffers for BSCREAD and BSCWRITE	41
Figure	5.	BSC Return Codes	57
Figure	6.	\$RJE Attention Requests	74
Figure	7.	Sample \$RJE Session (Part 1 of 2)	78
Figure	8.	Sample \$RJE Session (Part 2 of 2)	79
Figure	9.	System Status Data Set Sample Program	86
Figure	10.	TP Return Codes (Part 1 of 3)	102
Figure	11.	TP Return Codes (Part 2 of 3)	103
Figure	12.	TP Return Codes (Part 3 of 3)	104
Figure	13.	Remote Management Utility Record Types	209
Figure	14.	Remote Management Utility Record Scheme	210
Figure	15.	Communications Flow for the ALLOCATE Function	215
Figure	16.	Communications Flow for the DELETE Function	217
			219
Figure	18.	Communications Flow for the EXEC Function	222
Figure	19.	Communications Flow for the IDCHECK Function	224
Figure	20.	Logic Flow of a PASSTHRU Session	230
Figure	21.	Example of Passthru Records Received by Host	235
Figure	22.	Communications Flow for the PASSTHRU Function	
(Part	1 of	• 2)	241
Figure	23.	Communications Flow for the PASSTHRU Function	
(Part	2 of	<sup>:</sup> 2)	242
Figure	24.	Communications Flow for the RECEIVE Function	246
Figure	25.	Communications Flow for the SEND Function	250
Figure	26.	Communications Flow for the SHUTDOWN Function	253
Figure	27.	Communications Flow for the WRAP Function	255
Figure	28.	Error Handling by the Remote Management	
Utilit	: у		278
Figure	29.	Error Handling by the Host Program	279
Figure	30.	CDRRM Copy Code (Part 1 of 6)	292
Figure	31.	CDRRM Copy Code (Part 2 of 6)	293
			294
			295
			296
			297

The Event Driven Executive can be used to interact with a variety of terminals. Various techniques are available using the Event Driven Executive. These techniques support applications ranging from simple applications interacting with a single terminal to complex communications networks. This book contains both the information you need to understand which Event Driven Executive technique is best suited for your particular application and, the information needed to design your application using the technique selected.

#### These techniques are:

- Terminal Support
- Binary Synchronous Communication Access Method (BSCAM)
- Host Communications Facility
- Multiple Terminal Manager
- Remote Management Utility
  - Graphics
  - Utilities

# PART I - TECHNIQUE SELECTION CRITERIA

This part of the book lists and briefly describes the techniques supported by the Event Driven Executive. The information in this part of the book allows you to see what is available and to select the technique best suited for your particular application.

# CHAPTER 1. TECHNIQUES AVAILABLE USING THE EVENT DRIVEN EXEC-UTIVE

This chapter describes the techniques available using the Event Driven Executive. The information in this chapter will help you select the technique best suited to your application.

#### TERMINAL SUPPORT

This technique is the basic Event Driven Executive terminal support and should be selected if you are writing an Event Driven Language program which will interact with a single terminal. See Figure 1 on page 6 for supported terminals. Use of this facility will also allow the Event Driven Language program to interact with another Event Driven Language program. The interaction with another Event Driven Language program is known as virtual terminal support. For information on the virtual terminal support, refer to "Virtual Terminal Communications" in the System Guide.

Using the terminal support, you interact with the terminal in either field or line mode. If the terminal is a supported display, the interaction may also be in full screen mode. For information on the full screen mode support, refer to "Defining and Accessing Logical Screens" in the <u>System Guide</u>.

The following figure lists the devices and features which are supported by the Event Driven Executive.

Device (or equivalent) Attach Via Series/1 Controller/Adapter Feature Number IBM 2741 1610 IBM 4973 5630 IBM 4974 5620 IBM 4978 RPQ D02038 IBM 4979 3585 IBM 1610 Series/1 IBM 5100 1610 IBM 5110 1610 1610 or, 2091 with 2092 or, IBM 3101 2095 with 2096 or, 7850 ASCII terminal\* 1610 or, 2091 with 2092 or, 2095 with 2096 or, 7850 Graphics terminal\*\* 1560 \*Teletype¹ ASR 33/35 (TTY) or equivalent \*\*Tektronix<sup>2</sup> Model 4013 or equivalent 1560 - Integrated Digital Input/Output Non-Isolated 1610 - Asynchronous Communications Single Line Controller 2091 - Asynchronous Communications Eight Line Controller 2092 - Asynchronous Communications Four Line Adapter 2095 - Feature Programmable Eight Line Controller 2096 - Feature Programmable Four Line Adapter 3585 - 4979 Display Station Attachment 5620 - 4974 Printer Attachment 5630 - 4973 Line Printer Attachment 7850 - Teletypewriter Adapter RPQ D02038 - 4978 Display Station Attachment

Figure 1. Supported Devices and Features

Trademark of Teletype Corporation.

Trademark of Tektronix, Inc.

# Terminology for Supported Terminals

The following is a definition of terminologies used in describing Event Driven Executive supported terminals. This terminology is also used to describe the coding of the TERMINAL statement during system generation time which is discussed in the <a href="System Guide">System Guide</a>.

<u>Terminology</u> ASCII Terminal	Definition Any device which attaches via #7850, #1610, #2091 with #2092, or #2095 with #2096 adapters. (Teletypewriter, Asynchronous Single Line, Asynchronous Multiline, and Feature Programmable adapters respectively) and uses code type ASCII or EBASC.
ACCA Terminal	An ASCII terminal attached via #1610, #2091 with #2092, or #2095 with #2096 adapters.
Mirror Image	An ACCA terminal attached via #1610 or
ACCA Terminal	#2091 with #2092 using code type EBASC.
Real Image	An ACCA terminal attached via #2095
ACCA Terminal	with #2096 using code type ASCII.
ТТҮ	Any Teletype ASR 33/35 or compatible terminal attached via #7850 only.
2741 Terminal	A terminal attached via #1610 using code type CRSP or EBCD.
PROC	A terminal attached via #1610 using code type EBCDIC.

Note: Appendix A of this book contains all the code types mentioned in the previous text.

The following table shows the different device configurations.

Device Class	Code Type	Local/ Remote	Control/ Adapter	System Configuration Device Type	Device
Display	Graphics	Local	#1560	4013	4013
Display	ASCII	Remote	#2095 with #2096	ACCA	3101
Display	Mirror Image ASCII	Remote	#1610 or #2091 with #2092	ACCA	3101
Display	EBCDIC	Local	RPQ D02038	4978	4978
Display	EBCDIC	Local	#3585	4979	4979
Display	ASCII	Local	#7850	TTY	3101
Printer	ASCII	Local	#7850	TTY	Teletype
Printer	Real Image ASCII	Remote	#2095 with #2096	ACCA	Teletype
Printer	Mirror Image ASCII	Remote	#1610 or #2091 with #2092	ACCA	Teletype
Printer	CRSP	Remote	#1610	2741	2741
Printer	EBCD	Remote	#1610	2741	2741
Printer	EBCDIC	Local	#5620	4974	4974
Printer	EBCDIC	Local	#5630	4973	4973
Program	EBCDIC	Local	n/a	VIRT	Series/1
Program	EBCDIC	Remote	#1610	PROC	Series/1
Program	EBCD	Remote	#1610	PROC	5100/5110
Program	CRSP	Remote	#1610	PROC	5100/5110

#### BINARY SYNCHRONOUS COMMUNICATIONS ACCESS METHOD (BSCAM)

The binary synchronous communications access method technique should be selected when interacting with remotely connected terminals or CPUs using the binary synchronous communications facility. The remote terminals and CPUs may be any which support the BSC protocol. In order to use this technique, the connection must be via a BSC line. The Event Driven Language support allows you to write programs which send and receive data consistent with the BSC protocol on the line. This support also provides IBM utilities or, IBM supplied applications, which have general applicability. These utilities are 2780 and 3780 RJE emulators and aids for the debugging of programs which use binary synchronous communications.

#### HOST COMMUNICATIONS FACILITY

The Host Communications Facility allows you to send/receive data sets and background jobs to/from a host system. It requires the Host Communications Facility Installed Users Program (IUP) (5796-PGH) be installed on the host S/370 system. This technique provides Event Driven Language instructions and a utility (\$HCFUT1) to provide interactive communications between a S/370 host and remote Series/1 over a binary synchronous communications facility. The Host Communications Facility utilizes the BSCAM support to perform its functions.

## MULTIPLE TERMINAL MANAGER

The Multiple Terminal Manager support should be selected when the requirement is to support a transaction-oriented application. A transaction-oriented application is one which consists of several terminals, each of which may request concurrent interaction with one or more programs. The Multiple Terminal Manager manages the Series/1 storage area to reduce the amount of storage required to support interaction with more than one terminal by one program. High-level language support is provided.

# | REMOTE MANAGEMENT UTILITY

The Remote Management Utility support should be selected when the requirement is to provide remote Series/1 processing for a host computer. The Remote Management Utility provides a means of distributed processing on a remote Series/1, with little or no operator intervention required. The Remote Management Utility and the host communicate via a user-written host program over a BSC line using the BSCAM support of the Event Driven Executive.

#### **GRAPHICS**

This support should be selected when the application has a requirement for graphics support. This technique enables you to communicate with a Tektronix Model 4013 (or equivalent) terminal. The physical connection is via the #1560 adapter. In addition to the basic terminal support statements available, graphics-oriented Event Driven Language statements and IBM utilities are provided.

#### UTILITIES

Various IBM utilities are supplied to ease the burden of data transmission to/from interacting devices. These utilities are described in the appropriate sections and are:

 Terminal support (see <u>Utilities</u>, <u>Operator Commands</u>, <u>Program Preparation</u>, <u>Messages and Codes</u>)

> \$IMAGE \$FONT \$PFMAP \$TERMUT1 \$TERMUT2 \$TERMUT3

BSCAM (see Chapter 3)

\$PRT2780 \$BCSUT1 \$PRT3780 \$BSCUT2 \$RJE2780 \$RJE3780

Host Communications Facility (see Chapter 4)

\$HCFUT1

Graphics (see Chapter 7)

\$DIUTIL \$DICOMP \$DIINTR

# PART II - TECHNIQUE DESIGN INFORMATION

This part of the book describes in detail the different techniques supported by the Event Driven Executive. After you have selected the technique which best suits your application, you can design your application using the information provided in this part of the book.

The Event Driven Executive terminal support is designed to be as device independent as possible. With few exceptions, you need not be concerned with what type of device is being driven by terminal functions coded in the program. The same sequence of terminal output instructions, for instance, can be used to print data on a matrix or line printer, on a locally attached teletypewriter device, on a remote 2741 terminal, or to display the data on an electronic display screen device.

Terminals are defined in the system with the TERMINAL system configuration statement. This statement generates system control blocks and tables that contain the logical and physical variables required to operate the terminal.

The high degree of device independence is achieved in part by treating all terminals as though they were line printers, differing only in their page sizes (forms length) and margin settings, which are defined by TERMINAL statement operands. The support provides instructions allowing interactive communications between you and your application programs. See Figure 1 on page 6 for a list of supported terminals.

Generally, you can write terminal I/O functions in an application program without concern for the actual terminal being used. The default terminal to be used by the program is dynamically assigned by the supervisor to be the same terminal that was used to initially invoke the program. Therefore, the terminal assigned can vary from one program invocation to the next, with little or no program change. Utilizing the terminal instructions, any application program that contains no device dependent information can be operated in a compatible manner from any Event Driven Executive supported terminal.

Terminals can be referenced by symbolic name and accessed by any application program through appropriate instructions. Forms and screen format control can be dynamically changed within your program and the 4978/4979 screen can be copied to any designated hard copy terminal.

#### Terminal Operations

When a program is loaded from a terminal, that terminal is dynamically designated by the system as the terminal to be used by terminal I/O instructions in the program. Each terminal I/O instruction automatically has exclusive use of the terminal while executing, and can request extended control for multiple I/O operations.

If more than one task is using the terminal, terminal operations from different tasks could become interspersed. When this is not desirable, you can specify the ENQT (enqueue terminal) instruction to reserve the terminal for the exclusive use of a task, thereby preventing other tasks from using the terminal until the task issuing the ENQT releases it with the DEQT (dequeue terminal) instruction.

You can also use ENQT to gain exclusive control of any other terminal. The symbolic name of a terminal is the name coded on the label of the TERMINAL statement that defines the device. Coding a name in the label field of the TERMINAL statement during system configuration automatically defines the terminal to the system as a global resource that can be enqueued (ENQT) by other programs. Normally, an IOCB statement would be used to establish the connection between the ENQT and the TERMINAL statements at execution time.

Three symbolic terminal names are used by the supervisor for system utility programs:

- \$SYSLOG Names the system logging device or operator station, and must be defined in every system. In the starter supervisor, \$SYSLOG defines an IBM 4978 or an IBM 4979 Display Station.
- \$SYSLOGA Names the alternate system logging service. If unrecoverable errors prevent use of \$SYSLOG, the system will use the \$SYSLOGA terminal as the system logging device/operator station. If defined, this device should be a terminal with keyboard capability, not just a printer. The starter supervisor defines the \$SYSLOGA terminal as a teletypewriter device.
- \$SYSPRTR Names the system printer. If defined, the output from some system programs is directed to this device. The starter supervisor defines a 4974 matrix printer as the \$SYSPRTR device.

#### Terminal I/O Instructions

The Event Driven Language terminal I/O instructions are provided to control the input/output operations to terminals. These instructions are defined in the <a href="Language Reference">Language Reference</a> and are:

DEQT Releases a terminal from exclusive use

ENQT Acquires exclusive access to a terminal

ERASE Clears designated portions of static type screens

GETVALUE Reads one or more integer values that are entered by the terminal operator

PRINDATE Prints the date on the terminal

PRINTNUM Converts a floating-point variable or integer variable to printable form and writes it on the terminal, with an optional format specification

PRINTEXT Writes an alphameric text string to a terminal, with or without forms control

PRINTIME Prints the time of day on the terminal

QUESTION Prints a message and queries the operator for a Y (yes) or N (no) reply

RDCURSOR Acquires the cursor position of static screens

READTEXT Reads an alphameric text string from the terminal

TERMCTRL Controls device dependent features

#### Data Formatting Instructions

Data formatting instructions allow you to prepare formatted data for display on the terminals or printers attached to the Series/1. The capability is provided to format data in storage and then allow the program to decide the destination.

Use of the data formatting instructions FORMAT, GETEDIT, and PUTEDIT require that the user's object program be processed by the link edit program, \$LINK, in order to include the supervisor interface routines and the formatting routines which are supplied as object modules. Refer to the <u>Utilities, Operator Commands, Program Preparation, Messages and Codes</u> for the description of the autocall option of \$LINK, and information on the use of the "AUTO=\$AUTO,ASMLIB" option of \$LINK.

These instructions are defined in the <u>Language Reference</u> and are:

CONVTB Converts a binary value to an EBCDIC string.

CONVTD Converts an EBCDIC string to a binary value.

FORMAT Describes the conversion performed between internal and external representations of data items.

GETEDIT Receives data from a terminal using FORMAT.

PUTEDIT Sends data to a terminal using FORMAT.

#### Terminal Definition Functions

Two Event Driven Language statements are provided to define the type of terminal the program is connected to. These are:

TERMINAL A system configuration statement to define the existence of the terminal to the Event Driven Executive supervisor. This statement is defined in the <a href="System Guide">System Guide</a>.

IOCB Used in a program to define the variable attributes of a terminal, such as margins, and to supply the symbolic name of the TERMINAL statement supplied during system configuration. This statement is defined in the <a href="Language Reference">Language Reference</a>.

#### Interrupt Processing Functions

Normally a program would need to wait for an operator to respond to a request for input. This program wait capability is provided automatically by the READTEXT instruction or via the WAIT Event Driven Language instruction. The capability also exists to define asynchronous attention interrupt routines via the ATTNLIST instruction. When the Attention key is pressed on a terminal, the system will query the operator for a command. If this command is specified on the ATTNLIST statement, control is given to the appropriate program. These two instructions are defined in the Language Reference:

WAIT KEY - Wait for operator response.

ATTNLIST - Defines asynchronous attention interrupt routine.

See the <u>Language Reference</u> for a full discussion and sample programs illustrating the use of the terminal support technique.

#### Considerations for Feature #1610 or #2091 with #2092 Adapter

Devices attached via the #1610 controller or #2091 controller with #2092 adapters are supported by the standard terminal I/O instructions. The adapters operate in half-duplex mode and require special attention to the operating environment. Compared to the Event Driven Executive implementation of the #7850 adapter, the following differences are noted:

- Half-duplex mode
- No Series/1 Echo (must use Local Echo on terminal)
- Uses eight-bit data interchange code

The attached device may be used in a switched, leased, or direct connect environment. Each adapter feature has hardware jumpers that are used to customize the adapter to meet a variety of network configurations. Prior to defining the adapter to the Event Driven Executive via the TERMINAL statement, you should become familiar with these hardware jumpers. The Communications Feature Description should be referenced before actual connection of terminals or modems. Be sure the hardware is configured correctly prior to defining the software interface.

Some general rules for hardware jumpers are:

- For Direct Connect terminals:
  - Data Terminal Ready (DTR) is usually jumpered.
  - Request to Send (RTS); jumper only when Carrier Detect
     (CD) is not provided by terminal.
  - Carrier Detect (CD); jumper only when Request to Send (RTS) is provided by the terminal.
- For Leased Lines using modems:
  - Data Terminal Ready (DTR); jumper only when Event Driven Executive application programs do not control the modem.
  - Request to Send (RTS); jumper only if the modem provides a steady Clear to Send (CTS) signal.
  - Carrier Detect (CD); jumper only if the modem supports this feature.
- For Switched Lines using modems:
  - Data Terminal Ready (DTR); jumper only when Event Driven Executive application programs do not control the modem.
  - Request to Send (RTS); jumper only if the modem provides a steady Clear to Send (CTS) signal.
  - Carrier Detect (CD); jumper only if the modem supports this feature.

Speed range jumpers should be installed in accordance with instructions in the <u>Communications Feature Description</u>.

Once the hardware features have been properly defined, you may define the features to the Event Driven Executive system. The TERMINAL statement is used for this description. Additionally, the TERMCTRL statement has operands which allow the control of the modem. See the <a href="System Guide">System Guide</a> for information on the TERMCTRL statement and the <a href="Language Reference">Language Reference</a> for the TERMCTRL statement.

The TERMCTRL operands are as follows:

RING Waits until the Ring Indicator (RI) is presented to the Series/1 from the modem. No timeout is provided.

RINGT Waits until the Ring Indicator (RI) is presented to the Series/1 from the modem. If no Ring Indicator (RI) occurs after 60 seconds, then the instruction is terminated and an error condition is returned to the application program in the first word of the task control block (TCB).

ENABLE Activates Data Terminal Ready (DTR) if it is not already jumpered on and then waits for Data Set Ready (DSR) to be returned by the modem. No timeout is provided.

ENABLET Activates Data Terminal Ready (DTR) if it is not already jumpered on and then waits for Data Set Ready (DSR) to be returned by the modem. If Data Set Ready is not returned within 15 seconds, then the instruction is terminated and an error condition is returned to the application program in the first word of the TCB.

ENABLEA Provides the same function as ENABLE except that an answer tone is activated for 3 seconds following the activation of Data Set Ready (DSR). The modem must allow for the control of the answer tone.

ENABLEAT Provides the same function as ENABLET and ENABLEA combined.

DISABLE Disables Data Terminal Ready (DTR) if it is not jumpered on and waits for 15 seconds. Use this function to hang up the modem.

# Return Codes

After each I/O instruction issued by the Event Driven Executive application program, a return code is provided in the first word (taskname) of the TCB. These return codes have special meaning for terminals attached via #1610 controller, #2091 controller with #2092 adapters and #2095 controller with #2096 adapters.

-1	Successful completion.
Bit	Description
0 1-8 9-10 11 12 13 14-15	Unused ISB of last operation (I/O complete) Unused 1 if a write or control operation (I/O complete) Read operation (I/O complete) Unused Condition code +1 after I/O start (or) Condition code after I/O complete

Figure 2. Terminal I/O - ACCA Return Codes

If any error has occurred after I/O complete, then the cycle steal status information is also available at #CCBSTWO, #CCBSTW1 and #CCBSTW2. If the supervisor is mapped into your partition, you can obtain the three cycle steal status words by coding the following instructions:

PY PR	OGEQU
PY CC	BEQU
VE #1	, \$PRGCCB GET ADDRESS OF CCB
VE SA	VE,(#CCB-#CCBSTWO,#1),3 MOVE STATUS
TA 3F	'0'
	IPY CC

Refer to the <u>Communications Feature Description</u> for a detailed description of the Interrupt Status Byte (ISB) Condition Codes both after start I/O and after I/O complete as well as the meaning of the cycle steal status words 1, 2, and 3.

# Considerations for Feature #2095 with #2096 Adapter

The Event Driven Executive system includes support for the Feature Programmable Controller and Adapter. The #2095 controller with #2096 adapter has two modes of operation:

- Compatibility mode allows the substitution of #2095 controller with #2096 adapter for current asynchronous communication features (#1610 controller and #2091 controller with #2092 adapter using eight bit interchange code).
- EXIO mode provides access to the full command set.

In compatibility mode the difference between the #1610 controller and the #2091 controller with #2092 adapter is that the line code is ASCII. This is of particular importance during system configuration because the line control characters specified on the TERMINAL statement are not coded in mirror image, but in standard ASCII. The line code (CODTYPE) must also be specified as ASCII. Refer to the <u>System Guide</u> for details and a definition of mirror image.

# #7850 Teletypewriter Adapter

The most frequent use of the #7850 Teletypewriter Adapter support is to receive or send messages composed of ASCII character strings between the Series/1 and a teletypewriter terminal. The most common forms of such terminals are keyboard/printer and keyboard/CRT type display configurations. However, use of the terminal I/O instructions need not be limited to these types of terminals.

Devices are available from many vendors which are compatible with the physical transmission methods of the Series/1 Teletypewriter Adapter, for example, Isolated Contact sense, TTL, and EIA. Such devices include terminals which transmit only, or receive only, or transmit only in response to being polled for information. The devices may not have keyboards for information input but may acquire data from bar code scanners, analog or digital input features within the device, etc. The transmission code employed by these devices may be alphameric ASCII characters or may be any of the 256 possible 8-bit character combinations.

Proper use of the terminal I/O instructions enables your program to communicate with many such devices. For example, if the device attached to the #7850 Teletypewriter Adapter does not expect the data which it transmits to be returned by the Series/1 (usually returned for printing purposes), then the ECHO=NO parameter on the appropriate TERMINAL statement should be coded.

Further, the device data transmission to the Series/1 may include bit combinations which match the LINEDEL and CHARDEL parameter characters defined on the TERMINAL statement. To receive these characters as data in your program, the READTEXT instruction must specify the parameter XLATE=NO. XLATE=NO will permit the reception, as data, of any 8-bit pattern except for the carriage return (hexadecimal values OD or 8D). You may detect the reception of a carriage return character by performing the input operation as one or more READTEXT instructions, each of which specifies an input area that is one character in length. If the READTEXT operation completes with the received character count equal to zero, the character input was either an X'8D' or X'0D' value since reception of a carriage return terminates a READTEXT instruction without passing that character into your input area. There is no method available to distinguish between reception of X'OD' and X'8D' values.

Transmission of other than standard alphameric ASCII characters to a terminal is accomplished by specifying XLATE=NO on the PRINTEXT instruction. In this case, you must define the 8-bit values to be transmitted by means of DATA or DC instructions. The output data area must have the same format as is generated by a TEXT instruction.

## Special Considerations for the IBM 3101 in Character Mode

The IBM 3101 Display Terminal can be connected to the Series/1 via four attachments: the #7850 Teletypewriter Adapter, #1610 controller, #2091 controller with #2092 adapter, or #2095 controller with #2096 adapter. In the following discussion, all connections are direct, with no intervening modem. For a discussion of leased and switched lines using modems, refer to "Considerations for Feature #1610 or #2091 with #2092 Adapter" on page 17.

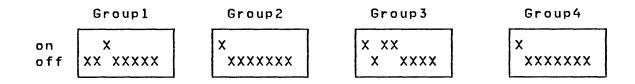
For attachment with the #7850 Teletypewriter Adapter, the #7850 input selection jumpers (see IBM Series/1 User's Attachment Manual, GA34-0033) may be set as follows:

			Input	Input
MSB		LSB	Selected	Interpreted as
0	1	0	EIA	Minus=datamark

MSB = Most Significant Bit LSB = Least Significant Bit

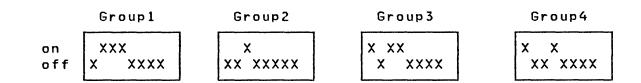
Also, the bit rate selection jumpers must match the 3101 setup switch settings.

A typical setup switch setting would be:



In the illustration above, the 3101 setup switch settings indicate 4800 bps. The #7850 bit rate selection jumpers would then also indicate 4800 bps. A bit rate of 110 bps would require that two stop bits be set in the 3101 setup switches instead of one as illustrated above.

For attachment via the #1610 controller or #2091 controller with #2092 adapter, or #2095 controller with #2096 adapter, the 3101 setup switches may be set as follows:



The jumpers for the #2091 controller with #2092 adapter should have Data Terminal Ready and Request to Send jumpered on. Also, the HIGH or LOW speed option must be jumpered to reflect the speed set in the 3101 setup switches. In the illustration above, the speed is 9600 bps. The RANGE and BITRATE operands on the TERMINAL configuration statement must also be compatible with the #2091 controller with #2092 adapter jumpers and 3101 setup switches.

The jumpers for the #2095 controller with the #2096 adapter should have Data Terminal Ready, Request to Send, and Receive Line (on = mark) jumpered on.

Finally, special consideration must be given to operator input and internal code representation. This is summarized in the following table.

		Character Generated		
		Device:	= ACCA	Device=TTY
Operator Function	Key on 3101	#1610 or #2091 with #2092	#2095 with #2096	#7850
		EBASC	ASCII	ASCII
ATTENTION	ESC followed by space bar	X'D9'	X'9B'	X'1B'
ENTER	<	X'B1'	X'8D'	X'OD'
BACKSPACE (character delete)	< (top row, not bottom row)	X'11'	X'88'	X'08'
LINE DELETE	DEL	X'FF'	X'FF'	X'7F'

Note that ECHO=NO or PROTECT=YES on the READTEXT statement (for suppression of input text) has no effect when the 3101 is attached via the #1610 controller or the #2091 controller with #2092 adapter, or the #2095 controller with #2096 adapter.

## | Special Considerations for the IBM 3101 in Block Mode

The IBM 3101 Model 2 may be operated in block mode under control of the Multiple Terminal Manager.

For attachment via the #1610 controller or #2091 controller with #2092 adapter, or #2095 controller with #2096 adapter, the 3101 setup switches may be set as follows:

	Group1	Group 2	Group3	Group4
on off	xxx x	xx xx	xxxxxxx	xxx

The jumper for the #2091 controller with #2092 adapter should have Data Terminal Ready and Request to Send jumpered on. Also, the HIGH or LOW speed option must be jumpered to reflect the speed set in the 3101 setup switches. In the illustration above, the speed is 2400 bps. The RANGE and BITRATE operands on the TERMINAL configuration statement must also be compatible with the #2091 controller with #2092 adapter jumpers and 3101 setup switches.

The jumpers for the #2095 controller with the #2096 adapter should have Data Terminal Ready, Request to Send, and Receive Line (on = mark) jumpered on.

Refer to the <u>System Guide</u> for sample TERMINAL statements and other system generation considerations.

Sample Terminal Support Program (1 of 3): The following example shows how to use the terminal support technique to print the IBM logo and the time and date.

```
PROGRAM START, 500, TERMERR = ERROR
SAMPLE
***********************
  NOTE THAT THE SUPERVISOR USED TO EXECUTE THIS
  SAMPLE PROGRAM MUST HAVE BEEN SYSGEN'ED TO INCLUDE TIMERS,
  $SYSLOGA, AND THE TARGET COMMUNICATIONS TERMINALS. THE
  NAME OF A TARGET TERMINAL IS THE LABEL USED ON THE TERMINAL *
  STATEMENT DESCRIBING IT.
**************************
                               TARGET TERMINAL IOCB
TERMX
        IOCB $SYSLOGA
        SPACE 2
START
        EQU *
* ASK OPERATOR FOR NAME OF TARGET TERMINAL. MOVE THAT NAME
* INTO THE 'TERMX' IOCB AND THEN 'ENQT' ON TERMX. THIS WILL
* ALLOCATE THAT TERMINAL TO THIS PROGRAM AND ALL TERMINAL I/O *
* INSTRUCTIONS WILL THEN BE ROUTED TO IT.
        READTEXT TNAME, 'ENTER 8 CHAR TERMINAL NAME:
        MOVE TERMX, TNAME, (8, BYTE) MOVE 8 CHARS TO IOCB
* DETERMINE THE LINE CONNECTION TYPE. IF SWITCHED, INQUIRE
* IF THE CPU IS THE CALLER OR THE ANSWERER.
       QUESTION 'als THE LINE CONNECTION SWITCHED? ', NO=XFER
       MOVEA LINETYPE, +SWITCHED INDICATE SWITCHED CONNECTION
       PRINTEXT 'a *** ANSWER THE FOLLOWING QUESTION, THEN'
       PRINTEXT ' PERFORM THE DIAL OPERATION *** '
       QUESTION 'als THE CPU THE CALLER? ', YES = XFER
       MOVEA DIALTYPE, + ANSWER INDICATE CPU WILL ANSWER
       ENQT TERMX
XFER
       IF (LINETYPE, EQ, +SWITCHED) SWITCHED CONNECTION?
         IF (DIALTYPE, EQ, + ANSWER) CPU TO ANSWER?
           TERMCTRL RING WAIT FOR RING INT. TO ANSWER
         ENDIF
         TERMCTRL ENABLE WAIT FOR DATA SET READY
       ENDIF
       EJECT
```

### Sample Program (2 of 3)

```
*-----*
* NOW THAT ALL TERMINAL I/O IS GOING TO
* THE TARGET TERMINAL:
                  1. PRINT IBM LOGO
                  2. PRINT DATE AND TIME
       EQU *
LOGO
       PRINTEXT LINE=1
       PRINTEXT LOGO1, SPACES=15, SKIP=4
       PRINTEXT LOGO2, SPACES = 15
       PRINTEXT LOGO3, SPACES = 15, SKIP=1
       PRINTEXT LOGO4, SPACES = 15
       PRINTEXT LOGO5, SPACES=15, SKIP=1
       PRINTEXT LOGO6, SPACES=15, SKIP=1
       PRINTEXT LOGO7, SPACES = 15, SKIP=1
       PRINTEXT LOGO8, SPACES = 15
       PRINTEXT LOGO9, SPACES = 15
       PRINTEXT SKIP=4
       SPACE
TIMES
       EQU
       PRINTEXT 'DATE = ',SPACES=5
       PRINDATE
       PRINTEXT 'TIME = ',SPACES=5
       PRINTIME
STOP
       EQU *
       GOTO ENDIT
                              END OF SAMPLE
       EJECT
     TERMINAL ERROR ROUTINE
       EQU *
ERROR
       MOVE RC, SAMPLE
                               SAVE THE ERROR CODE
                               RETURN TO SYSTEM CONSOLE
       DEQT
       PRINTEXT '3** UNRECOVERABLE TERMINAL ERROR OCCURRED ***
       PRINTEXT 'a THE ERROR CODE WAS '
       PRINTNUM RC, MODE=HEX
       PRINTEXT '0** SAMPLE IS TERMINATED **0'
       GOTO ENDIT
                              END THIS PROGRAM
RC
       DC
               F'0'
       EJECT
        END OF PROGRAM
```

## Sample Program (3 of 3)

```
ENDIT
       EQU *
       IF (LINETYPE, EQ, +SWITCHED) SWITCHED LINE CONNECTION?
         TERMCTRL DISABLE HANG UP IF SWITCHED CONNECTION
       ENDIF
                                END OF LINE CONNECTION TEST
       DEQT
                                RETURN TO CONSOLE USE
       PROGSTOP
       EJECT
       DATA AREA
       TEXT LENGTH=8 HOLDS NAME OF TARGET TERMINAL TEXT 'IIIIIIIII BBBBBBBB MM MMƏ' TEXT 'IIIIIIIII BBB BBBB MMM MMM'
TNAME
LOGO1 TEXT 'IIIIIIII
LOGO2 TEXT
LOGO3 TEXT '
                 ΙI
                            BBB BBBBB
                                            ' смммм мммм
       TEXT '
LOGO4
                  ΙI
                             BBBBBBBBBB
                                            MMMMMMMMM'
LOGOS TEXT '
                 ΙI
                            BBBBBBBB
                                            'MMMMMMMMM
LOGO6 TEXT '
                                            MM MMMM MM'
                 ΙI
                            BBBBBBBBBB
            · II
                                            MM MM
LOGO7 TEXT
                             BBB BBBB
                                                    ' GMM
                           BBB BBBB
             'IIIIIIIII
LOGO8
       TEXT
                                            MM
                                                    ' GMM
L0G09
     TEXT
             'IIIIIIIII
                            BBBBBBBB
                                            MM
                                                   MM'
DIALTYPE DATA F'-1'
                                DIAL CONNECTION TYPE:
        EQU -1
                                 -1 = CALL
CALL
ANSWER
        EQU 0
                                 0 = ANSWER
LINETYPE DATA F'0'
                                LINE CONNECTION TYPE:
SWITCHED EQU -1
                                -1 = SWITCHED
NONSW
        EQU 0
                                 0 = NON-SWITCHED
        ENDPROG
        END
```

## Interprocessor Communications

Using the #1610 Asynchronous Communication Single Line Controller Adapter feature with Event Driven Executive, processor to processor communication is available through the standard terminal interface. This mode of communication is specified by defining DEVICE=PROC on the TERMINAL statement. It allows connecting Series/1 to Series/1, Series/1 to IBM 5100 and IBM 5110 (using the Serial I/O feature), or Series/1 to any other processor capable of handling the required protocols. As with terminals, ATTENTION signals can be transmitted. The line protocol used by interprocessor communications is 2741 and is restricted to a single line ACCA feature #1610 per communication line to another processor. This provides a means to load or cancel programs, synchronize the action of tasks, and send and receive data to and from programs residing in remote processors. If CODTYPE=EBCDIC is defined on the TERMINAL statement, arbitrary binary data can be transmitted. The TER-MINAL statement is coded in your source statements for system generation, and is assembled together with DISK, SYSTEM, and to the other supervisor configuration statements. Refer section "System Configuration" in the System Guide detailed information.

### Hardware Preparation

In addition to defining the #1610 controller to the Event Driven Executive with the TERMINAL statement, you should set the hardware jumpers on the attachment according to the <u>IBM Series/1 Communications Feature Description</u>, GA34-0028.

<u>Note</u>: Interprocessor communication is restricted to the single line ACCA feature #1610.

For a direct processor interconnection:

- Data Terminal Ready (DTR) is jumpered
- Request To Send (RTS) is jumpered
- Low or High speed range is jumpered depending on the bit rate chosen (100 to 9600 baud).

Be sure to use the right cables for the type of attachments being interconnected. For a direct Series/1 to Series/1 connection, one side should use the Local Communication Cable (feature #2056) and the other should use the EIA Data Set cable (feature #2057) in order to interchange the Receive/Transmit lines; Data Set Ready (DSR)/Data Terminal Ready (DTR) and Request To Send (RTS)/Clear To Send (CTS). The #2056 cable allows attachment to a modem (male 25-pin type D connector); the #2057 cable allows attachment to a terminal (female 25-pin type D connector).

If only one cable type is available, the following lines of the 25-pin type D connectors have to be crossed:

Pin number (connector 1)	to	Pin number (connector 2)
1	Protective Ground XMT	1
2	Transmit Data (X or T)	3
3	Receive Data (REC)	2
4	Request to Send (RTS)	5
5	Clear to Send (CTS)	4
6	Data Set Ready (DSR)	20
7	Signal Ground	7
20	Data Terminal Ready (DTR)	6

For a Series/1 to IBM 5100 connection, the #2056 cable may be used.

#### Terminal Control Block (CCB)

When DEVICE=PROC is specified on the TERMINAL statement, the #1610 controller is defined as an interprocessor communications pipeline. The CODTYPE and CRDELAY parameters of the TERMINAL statement affect the protocol to be used. See "Modifications to the Protocol" on page 33. The BITRATE and RANGE parameters should be set in accordance with the hardware jumpers, matching the setting in the other processor. Also, the LINSIZE parameter should have the same value in both processors.

## Transmission Protocol

The length of a continuous message generated, for example, by a series of PRINTEXT commands on the sending side, might exceed the size of the receiving system buffer. Therefore the message is divided into records, which themselves may consist of subrecords (only for CODTYPE=EBCDIC).

A record corresponds to a line of text ended by a New Line (NL) character, the end of a message is defined as transition from Print to Read state and is indicated by an End of Transmission (EOT) character. Both messages and/or records may be empty; that is, contain no text (for example, in a transmission of SKIPs).

To a reading Event Driven Executive program, the received end characters are signalled as different return codes in the task code word. For the possible code types, the hexadecimal representation of the end characters is given, together with the corresponding return codes, in Figure 3.

	сорт	YPE=	
	EBCD/CRSP	EBCDIC	Return Codes
End of Transmission (EOT) End of Record (NL) End of Subrecord (EOSR)	1F 5B Not used	FDFF FEFF FCFF by	-2 -1 Handled y device support

Figure 3. Terminal I/O - Interprocessor Communications Return Codes

Note: For CODTYPE=EBCDIC, two characters are used to signal the respective end condition.

As in the IBM 2741 protocol, the beginning of a message (for example, the transition from Read to Print state) is indicated by transmission of an End of Address (EOA) character to the receiver (X'16' for EBCD/CRSP code. For EBCDIC code, see "CODTYPE=" on page 33.)

Before a message is sent, an EOT character indicating that the other side entered Read mode must be received. If this character has not been received as the end of the previous message, the device support waits the time period specified on the CRDELAY parameter, or the the default for this character. If it is not received, an error code (8) is returned to your program.

## Modifications to the Protocol

The communication protocol may be modified to satisfy special requirements by assigning the appropriate values to TERMINAL statement parameters. These options are discussed in "CRDELAY=" on page 33, and also in "CODTYPE=" on page 33.

#### CRDELAY=

- PROMPT, n The device support waits before every record (and subrecord) for the EOT prompt character. The time limit is n times 3.33 milliseconds, starting at the end of the previous operation. In response to the EOT, and also at the beginning of every record (and subrecord), an EOA character is sent.
- SP5100,n Identical to the PROMPT mode except that at End of Record, the two characters Line Feed and New Line (X'3B5B') are sent. This is necessary for communication with the IBM 5100 or IBM 5110 running APL or BASIC and using the Serial I/O feature.
- DELAY,n At the beginning of a message, the device support waits a maximum of one second for the EOT character(s). After each record a delay of n times 3.33 milliseconds is inserted. This mode might be used to simulate an 2741-like terminal for another processor.

#### CODTYPE=

CRSP With this option the #1610 controller is set to PTTC mode (see <u>Communications Feature Description</u>) and messages are translated via the CRSP conversion table (PTTC/correspondence code). The communication is restricted to characters, as PTTC mode allows only the transmission of bytes with the seven low-order bits of odd parity. Therefore, XLATE=NO should not be specified on PRINTEXT or READTEXT instructions.

EBCD Similar to CRSP, except that the EBCD conversion table is used. The EBCD option is recommended for connection to an IBM 5100 or IBM 5110 computer. The 6-bit code must be selected with the Serial I/O microprogram.

EBCDIC This option sets the #1610 controller to Eight Bit Coded Data Interchange mode with all change of direction codes equal to X'FF' Communications Feature Description). Special protocol provides for transparent exchange of arbitrary binary data. As there are no parity restrictions and only the code X'FF' is recognized as change of direction (indicating EOT, NL or EOSR), all bytes (especially all EBCDIC characters) other than X'FF' are transmitted "as is". Before a message or record is sent, it is scanned for a byte code (other than X'FF') not contained in it. This special code is sent as EOA and every occurring X'FF' in the message or record is replaced by it. On the receiving side, every EOA code is replaced by X'FF'. If a record is larger than 128 bytes, it is divided into appropriate subrecords (length < 128 bytes) to which the procedure can be applied.

<u>Note</u>: If CODTYPE=EBCDIC is used, arbitrary binary data may be transmitted.

#### CHAPTER 3. BINARY SYNCHRONOUS COMMUNICATIONS

The Event Driven Executive binary synchronous communications access method provides statements that allow you to write programs to send and receive data on a binary synchronous communications line. These statements are a part of the Event Driven Language and are coded in your application program. A general introduction to binary synchronous communications and details of the line protocol used by the Event Driven Executive may be found in <a href="General Information">General Information — Binary Synchronous Communications</a>, GA27-3004.

Series/1 binary synchronous communications closely parallels the System/370 and additional information on the subject of binary synchronous communications may be found in <u>IBM OS/VS</u>
<u>Basic Telecommunications Access Method (BTAM)</u>, GC27-6980.

Features of the binary synchronous communications access method provided with the Event Driven Executive are:

- Multiple line support
- Point-to-point leased line
- Point-to-point switched line (automatic answer, manual call and answer)
- Multipoint tributary station
- Multipoint master station
- Optional transparent mode
- Optional conversational mode

Hardware features and BSC protocol not supported by the Event Driven Executive are:

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

Throughout this chapter, the TYPE parameter is mentioned frequently, and refers to the TYPE parameter of the BSCLINE statement discussed in "BSCLINE" on page 42

For generation of BSC support into your Event Driven Executive supervisor, refer to the "System Configuration" section in the System Guide.

## Access Level

The Event Driven Executive BSC Access Method provides facilities at the READ/WRITE level. No control characters are inserted into or stripped from blocks of data in your buffer. However, all additional control sequences are managed by the access method in a manner transparent to the using program. You must ensure that the proper STX, DLE STX, ETX, and other control characters are contained in the output buffer. The single exception to this convention is the transmission of the DLE ETX or DLE ETB sequence to complete a transparent write, in which case these characters must not be included in the output buffer. On input, the buffer will contain all control characters received.

#### Conversational Operations

The BSC protocol provides a limited conversational response capability which is supported by the Event Driven Executive BSC Access Method. During conversational write operations, the response, which may be either an acknowledgement sequence or text, is read into a second buffer area specified by your program. Acknowledgement sequences are checked by the access method and error recovery is attempted when indicated. If text is received, a -2 return code is returned in lieu of the normal -1 and no error recovery is attempted.

Conversational writes may also be used to perform other special functions. For example, an IAM/WRU (I am/Who are you) explanation sequence can be transmitted by a calling station on a switched network using a conversational write.

#### Multipoint Operations

When the Series/1 is operating as the control station on a multipoint line (TYPE=MC), the access method handles the polling/selection requirements of initial operations via a poll sequence whose location address is specified in the BSCIOCB statement. A single poll/select is associated with each operation. A 3-second time-out is always enabled during poll/select operations regardless of the TIMEOUT parameter specifications.

When the Series/1 is operating as a tributary station on a multipoint line (TYPE=MT), the access method assumes that polling/selection has been established before a read/write initial operation is requested. The Read Poll operation monitors the line for receipt of a polling or selection sequence.

It assumes the BSC Adapter has been jumpered for multipoint tributary operation. Once the line has been polled/selected, your program should check the next operation request and issue appropriate read/write initial operation.

The initialization phase for multipoint operation is accomplished by the control station transmitting the following sequence:

NUL, EOT, PAD, NUL, (poll or selection address), ENQ<sup>3</sup>

This is the polling/selection sequence. The NUL,EOT,PAD,NUL<sup>3</sup> portion is generated by the access method. The (poll or selection address),ENQ<sup>3</sup> portion is supplied by you and referenced in the BSCIOCB. Generally this sequence consists of three bytes containing address,address,ENQ<sup>3</sup>. Refer to <u>General Information — Binary Synchronous Communications</u>, GA27-3004 for details.

#### Task Control

An implied wait is associated with each operation; that is, no immediate exit capability is provided. However, you may choose to attach a separate task to perform the operations in an asynchronous manner.

Sample programs are included at the end of this section which illustrate the most common communications operations.

Commas are for readability only and not part of the data stream.

# The Event Driven Language BSC Statements

The following text describes the Event Driven Language (BSC) statements and their syntax.

## **BSCCLOSE**

BSCCLOSE is used to free a binary synchronous line for use by other tasks. If the line is switched (TYPE=SM or SA), it will also drop Data Terminal Ready causing the line to be disconnected.

# **Syntax**

label BSCCLOSE bsciocb,ERROR=,P1=,P2=

Required: bsciocb Defaults: None Indexable: bsciocb

<u>Operands</u>	<u>Description</u>
label	The optional symbolic name of the BSCCLOSE statement.
bsciocb	The symbolic address or indexed location of the BSCIOCB statement to be associated with the close operation. Close processing uses this BSCIOCB to determine the address of the line to be closed.
ERROR=	The symbolic address of the next instruction to be executed if an error occurs while closing the line. If not specified, control will be returned to the next sequential instruction. In either case, the return code will reflect the results of the operation. See Figure 5 on page 57.
Pn=	The optional labels, P1 and P2 to be affixed to the bsciocb and ERROR operands, respectively.

### **BSCIOCB**

BSCIOCB is used to specify the line address and buffer(s) for BSCCLOSE, BSCOPEN, BSCREAD and BSCWRITE operations. BSCIOCB is a non-executable instruction. The first word of the BSCIOCB is also used to return auxiliary information about the ending status of the operation.

If variable-length records are to be written, the length field (length1 operand) must specify the actual length of the message to be written. The value specified in the length field should be reset to the buffer length before issuing a READ. Figure 4 on page 41 lists the number of buffers required by each type of BSCREAD and BSCWRITE statement.

## <u>Syntax</u>

label BSCIOCB lineaddr,buffer1,length1,buffer2,

length2,pollseq,pollsize,P1=,P2=,

P3=,P4=,P5=,P6=,P7=

Required: lineaddr

Defaults: None

Indexable: Not applicable

#### Operands Description

label The symb

The symbolic name of the BSCIOCB for reference in a BSCCLOSE, BSCOPEN, BSCREAD, or BSCWRITE operation. Label may also be used by other instructions to reference the auxiliary information returned in the first word of the BSCIOCB. This word will contain:

- After successful receipt of text, the address of the last character received.
- For all other conditions, the Interrupt Status
   Word from the Series/1 BSC Adapter.

lineaddr The hardware address, in hexadecimal form, of the line on which to perform the operation.

buffer1 The address in storage of the first buffer to be used in an operation. This buffer is located in the target Address Space as defined by \$TCBADS.

length1 The length, in bytes, of the first buffer.

buffer2 The address in storage of the second buffer to be used in an operation. This buffer is located in the target Address Space as defined by \$TCBADS.

length2 The length, in bytes, of the second buffer.

pollseq The address in storage of the poll or selection sequence to be used in a multipoint control line initial operation.

pollsize The length, in bytes, of the poll or selection sequence.

Pn= The optional labels to be affixed to the lineaddr, buffer1, length1, buffer2, length2, pollseq, and pollsize operands, respectively.

Note: The polling and selection sequences, consisting of from one to seven characters, are followed by: ENQ,(Read or Write Initial)<sup>4</sup>. Specific sequences for a given device may be found in the device component description manual. Generally a 3-byte pollsize is sufficient for a sequence of address, address, ENQ<sup>4</sup> between Series/1 processors. The actual sequence is determined by the device type tributary.

Commas are for readability only and are not part of the data stream.

Read type	Number of buffers	Write type	Number of buffers	
С	1	С	1	
D	0	CA	2	
Ε	1	CVX	2	
I	1	cx	1	
P	1	CXB	1	
Q	0	D	0	
R	1	E	0	
U	1	EX	0	
		Ī	1	
		ΙV	2	
		IVX	2	
		IX	1	
		IXB	1	
		Q	1	
		N	υ Τ	
			U	
		U	ī	
		UX	2	

Figure 4. Required Buffers for BSCREAD and BSCWRITE

#### **BSCLINE**

The BSCLINE statement is coded as part of your supervisor configuration. See "System Configuration" in the <u>System Guide</u>. BSCLINE defines the binary synchronous lines to be supported in the generated system. One BSCLINE statement is required for each line to be referenced by programs using the Binary Synchronous Communications Access Method. All BSCLINE statements must be grouped together with the last BSCLINE statement including an END=YES specification.

#### <u>Syntax</u>

blank BSCLINE ADDRESS=, TYPE=, RETRIES=, MC=, END=

Required: None

Defaults: ADDRESS=9, TYPE=PT, RETRIES=6, MC=NO, END=NO

Indexable: Not Applicable

### Operands Description

ADDRESS= The hardware address (in hexadecimal) of the line.

### TYPE=

PT (Point-to-Point) - The line is a point-to-point (non-switched) line with a single remote station. The adapter should be jumpered with DTR permanently enabled.

SM (Switch Manual) - The line is on a switched network and connection will be established manually by the operator. The adapter should be jumpered for switched line operation and DTR should not be permanently enabled.

SA (Switched Auto Answer) - The line is on a switched network and calls should be answered automatically by the BSC Access Method (during BSCOPEN). The adapter should be jumpered for switched line operation and DTR should not be permanently enabled.

MC (Multipoint Control) - The Series/1 is the controlling station on a multipoint line. The adapter should be jumpered with DTR permanently enabled and multipoint line should not be jumpered.

MT (Multipoint Tributary) - The Series/1 is a tributary station on a multipoint line. The adapter should be jumpered for multipoint tributary operation with DTR permanently enabled.

RETRIES=

The number of attempts which should be made to recover from common error conditions before posting a permanent error.

MC=

NO - The binary synchronous adapter located at the address specified on the ADDRESS operand is either a medium speed, single line feature card or a high speed, single line feature card.

YES - The binary synchronous adapter located at the address specified on the ADDRESS operand is part of a multiline controller feature configuration. When generating supervisors using multiline controller attachments, note the following:

- The character string YES must be specified. Any other character string will be equivalent to NO.
- All multiline feature cards must start at a base address ending with either X'0' or X'8'. A BSCLINE statement must exist for the line at this base address if any of the other lines of the multiline attachment are to be used.

END=

YES, for the last BSCLINE statement in the system definition module.

## Examples:

BSCLINE ADDRESS=28, TYPE=PT, RETRIES=10, MC=NO
BSCLINE ADDRESS=30, TYPE=SM, RETRIES=2, MC=YES, END=YES

#### **BSCOPEN**

BSCOPEN is used to prepare a binary synchronous line for use by a task. It first enqueues on the line and then prepares it for interrupts. If the line is switched manual (TYPE=SM), it will also raise Data Terminal Ready and wait up to two minutes for the telephone connection to be established. If the line is switched auto-answer (TYPE=SA), it will wait indefinitely for the ring interrupt and then raise Data Terminal Ready.

### Syntax

label BSCOPEN bsciocb, ERROR=, P1=, P2=

Required: bsciocb Defaults: None Indexable: bsciocb

Operands

Description

The optional symbolic name of the BSCOPEN instruction.

bsciocb The symbolic address or indexed location of the BSCIOCB statement to be associated with the open operation. Open processing uses this BSCIOCB to

ERROR= The symbolic address of the next instruction to be executed if an error occurs while opening the line. If not specified, control will be returned to the next sequential instruction. In either case, the return code will reflect the results of the operation.

determine the address of the line to be opened.

Pn= The optional labels to be affixed to the bsciocb and ERROR operands, respectively.

Note: BSCOPEN assumes that point-to-point lines will be jumpered with Data Terminal Ready (DTR) permanently set on.

#### **BSCREAD**

BSCREAD is used to read data from a binary synchronous line. If the read is successful, the first word of the associated BSCIOCB will contain the address of the last character read.

#### Syntax

<u>Operands</u>

label BSCREAD type,bsciocb,ERROR=,END=,

TIMEOUT=, P1=, P2=, P3=

Required: type,bsciocb
Defaults: TIMEOUT=YES
Indexable: bsciocb

Description

label The optional symbolic name of the BSCREAD statement. The type of read operation to be performed. See type "BSCREAD Types" on page 46 for a description of each bsciocb The symbolic address or indexed location of the BSCIOCB statement to be associated with the read operation. ERROR= The symbolic address of the next instruction to be executed if an error (return codes 10 through 99) is encountered. If not specified, control will be returned to the next sequential instruction. In either case, the return code will reflect the results. END= The symbolic address of the next instruction to be executed if an ending condition (return codes 1 through 6) is encountered. If not specified, control will be returned to the next sequential instruction. In either case, the return code will reflect the results.

TIMEOUT YES - The access method will enable a 3-second time-out during receive operations. If data is not received within this interval, a time-out error will occur. The appropriate retry procedure will then be attempted up to the limit specified in the RETRIES parameter of the BSCLINE statement defining this line. For initial type reads, the time-out may occur both when attempting to establish the correct initial sequence and during the subsequent read of the first record.

NO - The access method will disable the 3-second time-out during all receive operations.

Pn= The optional labels to be affixed to the bsciocb, ERROR, and END operands, respectively.

#### BSCREAD Types

The eight types of read operations to binary synchronous lines are:

C - Read Continue

D - Read Delay

E - Read End

I - Read Initial

P - Read Poll

Q - Read Inquiry

R - Read Repeat

U - Read User

### Type Operation

C Read Continue - Used to read subsequent blocks of data after an initial block is received via a Read Initial.

Read Continue writes a positive response and reads a message block:

- Write ACK-0 (X'1070') or ACK-1(X'1061')
- Read Text The text received is either message text or an EOT (X'37')

D Read Delay - Used to acknowledge correct receipt of a block of data and to request that the transmitting station wait before sending the next block. Multiple Read Delays may be issued before resuming transmission of data via a Read Continue.

Read Delay writes a WACK sequence and checks for the proper ENQ response:

- 1. Write WACK (X'106B')
- 2. Read ENQ (X'2D')
- E Read End Used to acknowledge correct receipt of a block of data and to request that the transmitting station stop sending data. Only one Read End should be issued during a single transmission and Read Continues should then be issued until EOT is actually received.

Read End writes an RVI sequence and reads a message block:

- 1. Write RVI (X'107C')
- Read Text The text received is either message text or an EOT (X'37')
- I Read Initial Used to read the first block of data in a transmission. After a successful Read Initial, Read Continues should be issued until EOT is received.

Point-to-point operation (TYPE=PT,SA,SM).

A Read Initial monitors the line for an ENQ sent by the transmitting station, writes a positive response (ACK-O), and reads the message block that follows:

- Read ENQ (X'2D')
- 2. Write ACK-0 (X'1070')
- 3. Read message text

Multipoint operation controller operation (TYPE=MC).

Read Initial polls a tributary station and if the response to polling is positive, reads the message text.

- 1. Write EOT (X'37')
- 2. Write polling sequence from address location specified in BSCIOCB
- 3. Read message text

Multipoint operation tributary operation (TYPE=MT).

Read Initial writes a positive response (ACK-O), and reads the message block that follows.

- 1. Write ACK-0 (X'1070')
- 2. Read message text
- P Read Poll Used to read the polling/selection sequence received when the Series/1 is acting as a tributary station on a multipoint line (TYPE=MT). Upon successful completion, the specified buffer will contain the sequence received starting with the second station (control unit) address character. The content of the received data stream, including control characters is not checked by the access method. Once polled/selected, your program should check the next operation requested and issue the appropriate Read/Write Initial Operation.
- Read Inquiry Used to read an ENQ character. Read Inquiry will return an invalid sequence error if ENQ or EOT is not received. If EOT is received, the END= exit will be taken if specified.
  - 1. Read ENQ (X'2D')
- Read Repeat Used to request retransmission of the last block of data following an unsuccessful read. The Read statements retry most common errors up to the limit of the RETRIES operand of the BSCLINE statement; however, Read Repeat may be used to attempt further recovery depending on the actual error encountered.

Read Repeat writes a negative response to the remote station and reads a message block:

- 1. Write NAK (X'3D')
- 2. Read Text
- W Read User Used in special situations to simply receive data. No associated write operation is performed by the access method, the data is not checked, and no error recovery is attempted.

#### BSCWRITE

BSCWRITE is used to write data to a binary synchronous line.

## <u>Syntax</u>

label BSCWRITE type,bsciocb,ERROR=,END=,CHECK=,

P1=,P2=,P3=

Required: type,bsciocb Defaults: CHECK=YES Indexable: bsciocb

<u>Operands</u>	Description
label	The optional symbolic name of the BSCWRITE statement.
type	The type of write operation to be performed. See "BSCWRITE Types" on page 50 for a description of each type.
bsciocb	The symbolic address or indexed location of the BSCIOCB statement to be associated with the write operation.
ERROR=	The symbolic address of the next instruction to be executed if an error (return codes 10 thru 99) is encountered. If not specified, control will be returned to the next sequential instruction. In either case, the return code will reflect the results.
END=	The symbolic address of the next instruction to be executed if an ending condition (return codes 1 through 6) is encountered. If not specified, control will be returned to the next sequential instruction. In either case, the return code will reflect the results.
CHECK=	YES - Valid only for type CV or CVX. Normal checking of the response occurs.
	NO - The response is not checked for protocol valid-

to Write User and Read User.

ity. This provides a chained write to read similar

Pn= The optional labels to be affixed to the bsciocb, ERROR=, and END= operands, respectively.

#### **BSCWRITE Types**

Seventeen types of write operations can be issued to a binary synchronous communications line. They are:

C - Write Continue

CV - Write Continue Conversational

CVX - Write Continue Conversational Transparent

CX - Write Continue Transparent

CXB - Write Continue Transparent Block

D - Write Delay

E - Write End

EX - Write End Transparent

I - Write Initial

IV - Write Initial Conversational

IVX - Write Initial Conversational Transparent

IX - Write Initial Transparent

IXB - Write Initial Transparent Block

Q - Write Inquiry

N - Write NAK

U - Write User

UX - Write User Transparent

### Type Operation

Write Continue - Used to write subsequent blocks of data after an initial block is written via a Write Initial.

Write Continue writes message text and reads a response from the receiving station.

- 1. Write Text
- 2. Read Response
- CV Write Continue Conversational Used to write subsequent blocks of data in conversational mode.

Write Continue Conversational writes message text and reads a response into your buffer. Acknowledgement sequences are checked by the access method and error recovery is attempted when indicated. If text is received, a -2 return code is returned in lieu of the normal -1.

1. Write Text

- Read Response/Text
- CVX Write Continue Conversational Transparent Used to write subsequent blocks of transparent data in conversational mode.

Write Continue Conversational Transparent writes message text and the ending sequence, DLE ETX, and reads a response into your buffer. Acknowledgement sequences are checked by the access method and error recovery is attempted when indicated. If text is received, a -2 return code is returned in lieu of the normal -1.

- 1. Write Text
- 2. Write DLE ETX (X'1003')
- 3. Read Response/Text
- CX Write Continue Transparent Used to write subsequent blocks of transparent data after an initial block is written.

Write Continue Transparent writes message text and the ending characters, DLE ETX, that must follow transparent data and reads a response from the receiving station.

- 1. Write Text
- 2. Write DLE ETX (X'1003')
- 3. Read Response
- CXB Write Continue Transparent Block Used to write subsequent blocks of transparent data after an initial block is written. This operation is the same as BSCWRITE type CX except ETB is used instead of ETX as the ending character.

Write Continue Transparent Block writes message text and the ending characters DLE ETB, that must follow transparent data, and reads a response from the receiving station.

- 1. Write Text
- 2. Write DLE ETB (X'1026')
- 3. Read Response
- D Write Delay Used to inform the remote station that the transmission of the next block of data will be delayed. Multiple Write Delays may be issued before transmission of data is resumed.

Write Delay writes a temporary text delay (TTD) sequence to the receiving station and reads a NAK response. The purpose of this operation is to inform the receiving station of a TTD before resuming transmission of message blocks.

- 1. Write TTD (X'022D')
- 2. Read NAK (X'3D')
- E Write End Used to inform the remote station that the previous block of data was the last of this transmission. Write End writes an EOT:
  - 1. Write EOT (X'37)
- Write End Transparent Used to write a transparent EOT (DLE EOT). This sequence is most commonly used to notify the receiving station on a switched line that the transmitting station is disconnecting from the line. Write End Transparent writes DLE EOT:
  - 1. Write DLE EDT (X'1037')
- Write Initial Used to write the first block of data in a transmission. Write Initial first establishes the correct initial sequence (depending on the type of line), and then writes the first block and checks the response.
  - Point-to-point Operation (TYPE=PT,SA,SM)

Write Initial writes an ENQ to gain use of the line, reads positive response (ACK-O), writes the message text and reads the response to the text:

- 1. Write ENQ (X'2D')
- 2. Read ACK-0 (X'1070')
- 3. Write Message text
- 4. Read Response
- Multipoint Operation Controller Mode (TYPE=MC)

Write Initial selects a tributary station and if the response to selection is positive, writes message text, then reads the response:

- 1. Write EOT (X'37')
- Write selection sequence where location address is specified in BSCIOCB

- 3. Read ACK-0 (X'1070')
- 4. Write Message Text
- 5. Read Response

Multipoint Operation Tributary Mode (TYPE=MT)

Write Initial writes message text and reads a response from the controller station:

- Write Message Text
- 2. Read Response
- IV Write Initial Conversational Used to write the first block of data of a transmission in conversational mode.

Write Initial Conversational establishes the correct initial sequence (depending on the type of line), writes the first block of message text and reads a response into your buffer. Acknowledgement sequences are checked by the access method and error recovery is attempted when indicated. If text is received, a -2 return code is returned in lieu of the normal -1.

- Point-to-point Operation (TYPE=PT,SA,SM)
  - 1. Write ENQ (X'2D')
  - 2. Read ACK-0 (X'1070')
  - 3. Write Message Text
  - 4. Read Response Text
- Multipoint Operation Controller Mode (TYPE=MC)
  - 1. Write EOT (X'37')
  - 2. Write selection sequence found in BSCIOCB
  - 3. Read ACK-0 (X'1070')
  - 4. Write Message Text
  - 5. Read Response Text
- Multipoint Operation Tributary Mode (TYPE=MT)
  - 1. Write Message Text
  - 2. Read Response Text

IVX Write Initial Conversational Transparent - Used to write the first block of transparent data of a transmission in conversational mode.

Write Initial Conversational Transparent first establishes the correct initial sequence (depending on the type of line), writes the first block of message text and the ending characters, DLE ETX, that must follow transparent data and reads a response into your buffer. Acknowledgement sequences are checked by the access method and error recovery is attempted when indicated. If text is received, a -2 return code is returned in lieu of the normal -1.

- Point-to-point Operation (TYPE=PT,SA,SM)
  - 1. Write ENQ (X'2D')
  - 2. Read ACK-0 (X'1070')
  - 3. Write Message Text
  - 4. Write DLE ETX (X'1003')
  - 5. Read Response Text
- Multipoint Operation Controller Mode (TYPE=MC)
  - 1. Write EOT (X'37')
  - 2. Write selection sequence found in BSCIOCB
  - 3. Read ACK-0 (X'1070')
  - 4. Write Message Text
  - 5. Write DLE ETX (X'1003')
  - 6. Read Response Text
- Multipoint Operation Tributary Mode (TYPE=MT)
  - 1. Write Message Text
  - 2. Write DLE ETX (X'1003')
  - 3. Read Response Text
- Write Initial Transparent Used to write the first block of transparent data in a transmission. Write Initial Transparent first establishes the correct initial sequence (depending on the type of line), and then writes the first block of transparent data and checks the response. The block is terminated by the access method with DLE ETX.

- IXB Write Initial Transparent Block Same as IX except ETB is used instead of ETX as the ending character.
- Q Write Inquiry Used to write an ENQ character and to read the response (which may be either a control sequence or text) into your buffer. This sequence is most commonly used to request retransmission of the response to a message block. It also retries upon time-out.
  - 1. Write ENQ (X'2D')
  - 2. Read Response/Text
- N Write NAK Used to simply write a NAK character down the line. The most likely use of this operation is to respond "device not ready" to polling/selection when the Series/1 is operating as a tributary station on a multipoint line (TYPE=MT).
  - 1. Write NAK (X'3D')
- Write User Used in special situations to simply transmit a character stream. No associated read operation is performed by the access method, and no error recovery is attempted.
  - 1. Write buffer in BSCIOCB for length indicated.
- Write User Transparent Used in special situations to simply transmit a transparent character stream. No associated read operation is performed by the access method, and no error recovery is attempted.
  - Write the stream described by BSCIOCB buffer1/length1
  - Exit transparent write using the character pair described by BSCIOCB buffer2.

Note: The only valid character pairs which may be contained in buffer2 are DLE ETX, DLE ETB, or DLE ENQ.

#### Error Recovery

Each BSC operation results in a return code being returned in the calling task's TCB (may be referenced by the taskname). Figure 5 on page 57 describes these return codes. Three basic completion conditions are possible:

- Successful operation
- Ending sequence received (END=)
- Permanent error encountered (ERROR=)

The particular type of condition encountered determines which of two optional completion exits may be taken during a read or write operation.

The access method attempts to recover from common line errors, but hardware and specification errors are not retried. Your program is free to retry permanent errors, and under certain conditions such attempts may prove successful.

Auxiliary error information is returned in the first word of the BSCIOCB. After successful receipt of text, the address of the last character received is returned in this word. For all other conditions, the Interrupt Status Word (ISW) from the Series/1 BSC Adapter is returned.

Code		N = + = :
Code	Description	Notes
-2	Text received in conversational mode	
-1	Successful completion	
END=		
1	EOT received	
2	DLE EOT received	
3	Reversé interrupt received	
4	Forward abort received	
5	Remote station not ready (NAK received)	4
6	Remote station busy (WACK received)	4
ERROR=		
10	Timeout occurred	1
11	Unrecovered transmission error (BSC error)	1
12	Invalid sequence received	3
13	Invalid multi-point tributary write attemp	t 2
14	Disregard this block sequence received	1
15	Remote station busy (WACK received)	1
20	Wrong length record - long (No COD)	6
21	Wrong length record - short (write only)	2
22	Invalid buffer address	2
23	Buffer length zero	2
24	Undefined line address	2
25	Line not opened by calling task	2
30	Modem interface error	2
31	Hardware overrun	2
32	Hardware error	5
33	Unexpected ring interrupt	2
34	Invalid interrupt during auto-answer	
	attempt	2
35	Enable or disable DTR error	2
99	Access method error	2

Figure 5. BSC Return Codes

# Notes:

- 1. Retried up to the limit specified on the RETRIES operand of the BSCLINE definition.
- 2. Not retried.
- 3. Retried during write operations only when a wrong ACK is received following an ENQ request after timeout (indicating that no text had been received at the remote station).

- 4. Returned only during an initial sequence with no retry attempted.
- 5. Retried only after an unsuccessful start I/O attempt.
- 6. Retried only during read operations.

# Sample Program: Write Transparent

### SOURCE STATEMENT

```
PRINT NOGEN
WRITEX
         PROGRAM START
         BSCOPEN IOCB, ERROR = PRINTERR
START
RESTART
         BSCWRITE IX, IOCB
          ΙF
                (WRITEX, EQ, 10), GOTO, RESTART
          IF
                (WRITEX, NE, -1), GOTO, PRINTERR
         DO
                29, TIMES
           ADD
                 I,1
           CONVTB MSG#,I
           BSCWRITE CX, IOCB, ERROR = PRINTERR
          ENDDO
         BSCWRITE E, IOCB, ERROR = PRINTERR
         GOTO ALLDONE
PRINTERR MOVE ERRCODE, WRITEX
         PRINTEXT 'WRITE ERROR:', SKIP=1
         PRINTNUM ERRCODE
         BSCCLOSE IOCB
ALLDONE
         PROGSTOP
         BSCIOCB 19, BUFFER, 82
IOCB
BUFFER
                X'1002'
         DC
                CL74'TEST MESSAGE'
         DC
MSG#
         DC
                CL6'
                          1'
                F'1'
         DC
                F'0'
ERRCODE
         DC
        ENDPROG
        END
```

# Sample Program: Read Transparent

END

SOURCE STATEMENT

PRINT NOGEN PROGRAM START READX START ENQT \$SYSPRTR BSCOPEN IOCB, ERROR = PRINTERR BSCREAD I, IOCB RESTART ΙF (READX, EQ, 10), GOTO, RESTART IF (READX, NE, -1), GOTO, PRINTERR MOVE MSG, INPUT+2, (80, BYTE) PRINTIT PRINTEXT MSG, SKIP=1 BSCREAD C, IOCB, END=ALLDONE, ERROR=PRINTERR PRINTIT GOTO PRINTERR MOVE RETCODE, READX PRINTEXT ERRMSG, SKIP=1 PRINTNUM RETCODE BSCREAD R, IOCB, ERROR = ALLDONE, END = ALLDONE GOTO PRINTIT ALLDONE DEQT BSCCLOSE IOCB PROGSTOP IOCB BSCIOCB 29, INPUT, 83 INPUT CL83' ' DC MSG TEXT LENGTH=80 ERRMSG TEXT 'READ ERROR:' F'0' RETCODE DC ENDPROG

Note: The \$BSCUT2 utility contains many examples of the use of the Series/1 Event Driven Executive binary synchronous instructions. Examination of the source program for \$BSCUT2 should answer many questions on buffer content of both data to be transmitted and data received.

### Utility Programs (BSC)

This section describes the Event Driven Executive BSC utility programs and their syntax.

#### SBSCTRCE

The \$BSCTRCE utility program provides a means to trace the I/O activities on a given BSC line. \$BSCTRCE must be loaded in the same partition as the application program that is controlling the traced line. If loaded in any other partition, unpredictable results will occur. When loaded, \$BSCTRCE prompts for the disk or diskette file in which to place the trace output. \$BSCTRCE then prompts for the line number to be traced. The trace action is terminated by the attention command STOP. Since the output file is reused from the beginning whenever the end is reached, \$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. Multiple BSC lines may be traced concurrently with multiple loads of \$BSCTRCE using different trace files, for example:

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

...
> STOP

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

Trace File Record Format: The format of the records produced by \$BSCTRCE is shown below.

	cc	ISW	STATUS	DCB	LGTH	DATA	LAST4
0	] -	+2 -	+4 -	+10 +	+26 +	<b>⊦</b> 28 -	+252

\*CC Interrupt Condition Code on completion of the I/O.

\*ISW Interrupt Status Word on completion of the I/O.

\*STATUS The three status words of the BSC Adapter (produced

when bit 0 of the ISW is on.)

DCB The Device Control Block for the I/O.

LGTH The length of the data sent/received.

DATA The data in main storage following the I/O.

LAST4 The last 4 bytes of data if the data is longer than 227

bytes.

Note: \* These fields are zero when the DCB has been chained from the previous record's DCB.

#### \$BSCUT1

The \$BSCUT1 utility program formats binary synchronous trace files (see \$BSCTRCE utility description) to either \$SYSPRTR or a terminal. You may select the records of the trace file to dump. You will be prompted, as necessary, for information required by the functions of \$BSCUT1.

Following is a list of the available functions of \$BSCUT1, as obtained by using the ? command.

COMMAND(?): ?

CV - CHANGE VOLUME

DP - PRINT TRACE FILE ON PRINTER

DU - DUMP TRACE FILE ON TERMINAL

(CA WILL CANCEL)

EN - END PROGRAM

COMMAND (?):

Example: Dump trace file to your console

COMMAND (?): DU TRACE9

FIRST RECORD: 32 LAST RECORD: 33

DUMP OF TRACE FILE TRACE9 ON EDXO02

\*\*\*\*\* 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

\*\*\*\*\* RECORD 33 \*\*\*\*\* CONTINUATION OF CHAINED OPERATION

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

DATA LENGTH = 485 0227 615B F1F6 4BF5 F94B F3F4 40D1 D6C2 |../\$16.59.34 JOB| 4040 F4F2 F440 D7D9 F3F0 F1F6 F5F6 40C5 | 424 PR301656 E| 17 E7C5 C3E4 E3C9 D5C7 40D4 40D7 D9C9 D640 IXECUTING M PRIO 1 33 49 40F7 1E27 615B F1F6 4BF5 F94B F3F4 40D1 | 7../\$16.59.34 J| D6C2 4040 F4F2 F340 C8D8 F1F2 F1F6 F5F6 65 IOB 423 HQ1216561 40C5 E7C5 C3E4 E3C9 D5C7 40D4 40D7 D9C9 | EXECUTING M PRI 81 [0 7../\$16.59.34] 97 D640 40F7 1E27 615B F1F6 4BF5 F94B F3F4 113 40D1 D6C2 4040 F3F0 F040 C9E2 F0F3 F1F4 | JOB 300 IS0314| 129 F4F5 40C5 E7C5 C3E4 E3C9 D5C7 40E5 40D7 145 EXECUTING V P 145 D9C9 D640 40F5 1E27 615B F1F6 4BF5 F94B |RIO 5../\$16.59.| |34 JOB..48 #GSPE| 161 F3F4 40D1 D6C2 1D43 F4F8 407B C7E2 D7C5 177 FOF1 F040 D6D5 40D7 D9C9 D5E3 D9F2 4040 1010 ON PRINTR2 | 193 D7D9 C9D6 4040 F51E 2761 5BF1 F64B F5F9 |PRIO 5../\$16.59| 209 4BF3 F440 D1D6 C240 40F3 F2F0 40C6 C7F6 1.34 JOB 320 FG6 LAST 4 D4D5 1E26 IMN.. ١

DUMP COMPLETE ANOTHER AREA?

### SBSCUT2

The \$BSCUT2 utility program checks out the binary synchronous communications access method (BSCAM), the BSCLINE definitions generated in the executing supervisor, and the hardware customized jumper assignments in the adapters. Various BSCAM capabilities may be tested as follows:

- 1. Read and write both transparent and non-transparent data
- 2. Operate in limited conversational mode with both transparent and non-transparent data
- Operate as a master controller on a multipoint (multidrop) line to both poll and select tributaries (text written only for transparent data)
- 4. Operate as a tributary on a multipoint line and be polled and selected (text written only for transparent mode)

The primary purpose of this utility is to check out your system after installation, supervisor generation, and your tailored adapter assignments via the jumper options (device address, type such as PT, SM or SA, tributary address, etc.). Therefore it is essential to have this information available to run this program. For each selected function in \$BSCUT2, you will be prompted for the device (line) address, tributary address (if multipoint), record length, etc. Error messages will print if any discrepancies exist between the function being performed and the hardware assignments. These error codes are defined in this section.

Normal or successful exercising of any given function results in a test pattern message being printed or displayed on the selected output terminal. The output basically consists of:

 First section - Internal task identifier (for example READ for transparent and non-transparent reads), and optionally record number and record length.

Example:

Task READ entered RECORD NUMBER= 1 RECORD LENGTH= 80

2. Second line - Function identifier, record number, and alphabetic text string (A through Z) repeated to fill record length specified. The identifier and record number make up a 25-byte field and the remaining record length is filled by the alphabetic string. Therefore if you specified a record length of 80, the alphabetic string would consist of 55 characters (A through Z, A through Z, and ABC).

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

Following is a list of the available functions of \$BSCUT2 as obtained by using the ? command.

\$BSCUT2 74P,00:33:52:, LP=9400

COMMAND (?): ?

RWI ---- READ/WRITE - NONTRANSPARENT

RWIX --- READ/WRITE - TRANSPARENT

RWIMP -- READ/WRITE - MULTIDROP LINE NONTRANSPARENT

RWIXMP - READ/WRITE - MULTIDROP LINE TRANSPARENT

RI ---- READ - TRANSPARENT/NONTRANSPARENT

WI ---- WRITE - NONTRANSPARENT

WIX --- WRITE - TRANSPARENT

EN ---- END THE PROGRAM

CH ---- CHANGE HARDCOPY DEVICE

RWIVX -- READ/WRITE - TRANSPARENT CONVERSATIONAL

RWIV -- READ/WRITE - NONTRANSPARENT CONVERSATIONAL

\$BSCUT2 can be used to check out binary synchronous operations if at least two binary synchronous adapters are available on Series/1 processors and if a connection between the two adapters is made. If switched manual connections are used, \$BSCUT2 does not prompt you to make connection. This must be done once the \$BSCUT2 command has been issued and all questions have been answered.

Note: \$BSCUT2 contains many examples of the use of the Series/1 Event Driven Executive binary synchronous instructions. Examination of the source program for \$BSCUT2 should answer many questions on buffer content of both data to be transmitted and data received.

Following are explanations of each type of command for \$BSCUT2:

## RWI - Read/Write Non-transparent Data

This command writes non-transparent messages on line. Each message is numbered. The record length for write includes the control characters. The read task receives the messages, analyzes them, and prints them on the hardcopy device. The analysis includes transparent or non-transparent and record length received.

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

### Notes:

- 1. READ ADDRESS and WRITE ADDRESS refer to binary synchronous adapter channel address. 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. The other address can be invalid and the corresponding task 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.
- 2. RECL questions refer to the buffer size to be used and therefore the number of bytes transferred in one transmission over the binary synchronous line. The maximum buffer size permitted is 512 bytes. READ (RECL) should always be equal to or greater than WRITE or errors will occur.
- 3. NUMBER OF RECORDS determines the number of transmissions to be made before the test ends.
- 4. "Monitor" functions turn on a switch which allows each task to report its progress to the terminal. Thus TASK ENTERED, TASK EXITED messages and so on are written to the invoking terminal if the monitor function is enabled.

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

Same as "RWI - Read/Write Non-transparent Data" on page 66 except data transmitted by the WRITE task is transparent.

# RWIXMP - Read/Write Transparent, Multidrop Line

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

See notes under "RWI - Read/Write Non-transparent Data" on page 66. In this command, BUFFER LENGTH is equivalent to RECL.

The master controller (MC) at device address polls and selects all tributaries (MT) and sends and receives messages to them. Since each task both transmits and receives, successful operation requires the controller buffer length to equal all tributary buffer lengths. Values other than this can be entered to test access method error detection. Received messages are logged to the hardcopy device.

DEVICE ADDRESS for this command refers to binary synchronous adapter channel address. TRIBUTARY ADDRESS refers to the jumpered tributary address on each card.

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

If the test being performed is between two \$BSCUT2 programs then:

- Program 1 would use a valid MC device address and dummy tributaries (MT)
- Program 2 would use a dummy MC device address and valid tributaries (MT)
- 3. NUMBER OF TRIBUTARIES must be equal in both programs
- 4. LOOP COUNT must be equal in both programs.

## RI - Read Transparent/Non-transparent

COMMAND (?): RI

RI ---- READ - TRANSPARENT/NONTRANSPARENT

READ ADDRESS? 5A

READ RECL? 80

READ MONITOR? Y

See note under "WIX - Write Transparent" on page 69.

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

See note under "WIX - Write Transparent."

WIX - Write Transparent

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

Note: RI, WI, and WIX commands individually activate the tasks comprising RWI and RWIX. The Read task does not require NUMBER OF RECORDS since it will read either transparent or non-transparent data until EOT is received. This makes the Read task useful for monitoring any binary synchronous line sending data to the processor. For example, RI can receive data from \$RJE2780 or \$RJE3780 utilities operating in the same Series/1 or in another Series/1.

# EN - End \$BSCUT2 Program

COMMAND (?): EN

\$BSCUT2 ENDED AT 01:14:40

# CH - Change Hardcopy Device

COMMAND (?): CH

NEW HARDCOPY DEVICE? \$SYSLOGA

Note: If the hardcopy device entered is not defined, then the hardcopy output will come to the terminal which loaded \$BSCUT2.

# RWIVX - Read/Write Transparent Conversational

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

# RWIV - Read/Write Non-transparent Conversational

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

For RWIVX and RWIV commands, see Notes under "RWI - Read/Write Non-transparent Data" on page 66. In this command BUFFER LENGTH is equivalent to RECL.

RWIVX and RWIV test limited conversational operation in both transparent and non-transparent mode. The following is a description of the binary synchronous line transactions:

	READ TASK
ENQ> <acko (response)-<br="">Text&gt;</acko>	BSCREAD I
<pre><text (response)-="" (response)ack1=""></text></pre>	BSCWRITE CV(X)
<text< td=""><td>BSCWRITE CV(X)</td></text<>	BSCWRITE CV(X)
·	BSCREAD C
Text>	DOCKERD O
<text></text>	BSCWRITE CV(X)
	<acko (response)text=""> <text (response)-="" (response)ack1=""> <texttext (response)-=""> <acko (response)text=""> <text> <text></text></text></acko></texttext></text></acko>

This sequence continues until the NUMBER OF RECORDS count is satisfied.

## \$PRT2780 and \$PRT3780 Utility Programs

\$PRT2780 and \$PRT3780 are utility programs which will print the spool records produced by the \$RJE2780 and \$RJE3780 utilities. When these utilities are loaded, they prompt for the name of the spool file to be printed. The utility terminates upon reaching the end of the spool file. An initial option allows you to choose a printer other than \$SYSPRTR if desired.

#### Example:

> \$L \$PRT3780

DS1(NAME, VOLUME): ASMWORK

\$PRT3780 9P,00:02:44, LP= 8000

PRINT TO \$SYSPRTR? (Y OR N): Y

\$PRT3780 ENDED AT 00:03:05

Spooled data from a /\*DR HASP command during remote job entry session as printed out by above utility is:

\$19.28.14 RM74.RD1 \*\*\* INACTIVE \$19.28.14 RM74.PR1 \*\*\* INACTIVE \$19.28.14 RM74.PU1 \*\*\* INACTIVE \$19.28.14 RM75.RD1 \*\*\* INACTIVE \$19.28.14 RM75.PR1 \*\*\* INACTIVE \$19.28.14 RM75.PU1 \*\*\* INACTIVE

# \$RJE2780 and \$RJE3780 Utility Programs

\$RJE2780 is a utility program which can be used to interface with a System/360 or System/370 via remote job entry. It simulates an IBM 2780 having the following characteristics and features:

- Model 2 (Card reader, card punch, and printer)
- EBCDIC transparency
- Multiple record transmission
- 132-character print line
- Transparent punch output only
- No horizontal tab
- No tape controlled operations (except channel 1 as new page indicator)

\$RJE3780 is a utility program which can be used to interface with a System/360 or System/370 via remote job entry. It simulates an IBM 3780 having the following characteristics and features:

- 3780 with IBM 3781 Card Punch
- Compression for both input and output
- Vertical tab
- Transparent punch output only

\$RJE2780 and \$RJE3780 present the same interface to the following list of host RJE facilities:

- HASP or HASP V4
- JES2 or JES3
- RES
- VMRSCS

In the following pages, \$RJE refers both to \$RJE2780 and \$RJE3780.

The \$RJE utility is controlled by a set of attention requests. See Figure 6 on page 74.

ABORT	Stops transmission to or from the host
COMMAND	Sends a single card image to the host
END	Terminates execution of the utility
ENDSPOOL	Switches from spooling to direct printing
PRINTON	Defines the terminal name used for output
PUNCHO	Defines a disk or diskette file for punch output of object data
PUNCHS	Defines a disk or diskette file for punch output of source data
RESET	Reset function (use caution)
SPOOL	Defines a disk or diskette file for printer output and to commence spooling
SUBMIT	Sends a data stream to the host
SUBMITX	Sends a transparent data stream to the host

Figure 6. \$RJE Attention Requests

When the \$RJE utility is first loaded, it checks for the presence of only one BSC line specified in the supervisor. If true, the actual device address of the adapter is used as the default line address and a prompting message is suppressed. If more than one BSC line has been defined, it prompts for the RJE line address. Subsequent control operations are all performed using the attention request commands. Multiple copies of \$RJE can be loaded using different lines to the host. The spool facility can be used to avoid contention for a single printer. Figure 7 on page 78 and Figure 8 on page 79 show a sample \$RJE session.

### Attention Requests

ABORT: ABORT is used to stop a data transmission which is currently in process. During a SUBMIT or SUBMITX operation, normal end-of-file is transmitted to the host following the current block. During receive operations, EOT is returned instead of a normal acknowledgement and data then continues to be received until the host sends EOT. Depending on the operation of the host RJE system, this can result in suspension of print or punch output and a pause during which the host will receive

input. Since the pause for input by the host may be short, any desired commands (for example, to submit another job, cancel the current output, hold a job, or display status) should be entered before the ABORT command. This command simulates pressing STOP on a 2780 while printing or punching, CARRIAGE STOP on a 3780 printer while printing, or STOP on a 3781 punch while punching.

COMMAND: COMMAND is used to send a single card image record to the host. The most common use of this capability will be to send control commands and information requests to the host; for example, a HASP /\*\$DA command.

Upon entering the COMMAND attention request, you are asked to enter the command to be sent.

END: END is used to terminate the \$RJE utility program.

<u>ENDSPOOL</u>: ENDSPOOL is used to terminate the spooling of printer output (see SPOOL command). If a print data stream is being received and spooled when this command is entered, spooling will continue until the end of the data stream. Subsequent print data streams will then be printed on the defined printer.

<u>PRINTON</u>: PRINTON is used to define the name of the terminal to be used for print output. If not specified, \$SYSPRTR is assumed.

PUNCHS and PUNCHO: PUNCHS and PUNCHO are used to define a disk or diskette file to be used to receive punch data from the host. Card image punch data streams can be written to disk in two different formats: source (S) or object (O). Source format will produce two 80-byte card image records per 256-byte disk record with the second card starting at byte location 129. Object format will produce three 80-byte contiguous card image records per 256-byte disk record with the last 16 bytes set to hexadecimal zeros. The punch specification is automatically reset at the completion of each punch data stream so that multiple punch data streams can be separated into different output data sets by issuing another PUNCHS or PUNCHO command.

Upon entering the PUNCHS or PUNCHO attention request, you will be queried for the name and volume of the file to be used for punch output. If volume is not specified, the IPL volume is assumed. The file name and volume can also be specified as part of the PUNCHS or PUNCHO command, for example:

PUNCHS PUNCHOUT, EDXOO1

\$RJE examines the first cards received from the host and disregards those containing a X'6A' in columns 1, 10, and 11 (indicating a HASP punch header card). \$RJE must be modified by you to purge other than HASP punch header cards.

<u>RESET</u>: RESET is used to reset functions that have <u>not</u> started operation in \$RJE (for example buffered command images that have not yet been sent to the host, SUBMIT files that have not yet started transmission). RESET should be used with caution. If RESET is used, once a function is in process or if use of RESET overlaps a function initiation sequence, unpredictable results may occur. RESET conditionally prompts you with the following:

ENTER RESET TYPE (CO,SU,SP,PU):

CO - COMMAND function

SU - SUBMIT(X) function

SP - SPOOL function

PU - PUNCH(S or O) function

SPOOL: SPOOL is used to define a disk or diskette file to be used to receive printer data from the host. If not specified, \$RJE will print received data directly to the printer. Once specified, all printer output will be spooled until an ENDSPOOL command is issued. The utility programs \$PRT2780 or \$PRT3780 can be used to print the contents of a spool file produced by \$RJE2780 or \$RJE3780, respectively.

Upon entering the SPOOL attention request, you will be prompted for the name and volume of the disk or diskette file to be used for printer output. If volume is not specified, the IPL volume is assumed. The space allocated to this file must be at least equal in size (256-byte records) to the number of print lines to be spooled and there must be an even number of records in the spool file. Once the spool file is full, the output reverts to the defined printer. The spool file name and volume may also be entered with the SPOOL command, for example:

SPOOL SPOOLFLE, WRKLIB

SUBMIT and SUBMITX: SUBMIT is used to define and send a data stream to the host. SUBMITX is used to define and send a transparent data stream to the host. Multiple disk or diskette files may be sent using the /\*CONCAT statement in the data stream itself. The files must be in the same format as that produced by the \$EDIT1N and \$FSEDIT utility programs (for example, two 80-byte card image records per 256-byte disk or diskette record with the second card beginning at byte location 129). Two command statements within the data stream are recognized by \$RJE and are not transmitted to the host:

- 1. /\*END signifies the end of the data stream to be sent.
- /\*CONCAT filename, volume signifies that the data stream
  is to be continued using the file specified. If volume is
  not specified, the IPL volume is assumed. Any number of
  files may be concatenated into one data stream.

Upon entering the SUBMIT or SUBMITX attention request, you will be queried for the name and volume of the file to be sent to the host. If volume is not specified, the IPL volume is assumed. The submit file name and volume may also be entered with the SUBMIT or SUBMITX command, for example:

SUBMITX MYJOB, WRKLIB

```
> $L $RJE2780
$RJE2780
           35P,00:00:00, LP= 7C00
ENTER RJE LINE ADDRESS IN HEX: 5F
DIAL HOST
HOST CONNECTION ESTABLISHED
> COMMAND
ENTER COMMAND
/*SIGNON
               REMOTEXX
COMMAND READY TO SEND
COMMAND SENT
> PUNCHO
ENTER PUNCH FILE NAME (NAME, VOLUME): PCHOUT01, EDX002
PUNCH FILE DEFINED
> SUBMIT
ENTER SUBMIT FILE NAME (NAME, VOLUME): RJEJOBO1, EDX002
SUBMIT FILE READY TO SEND
FILE TRANSMISSION STARTED
FILE TRANSMISSION COMPLETED
> COMMAND
ENTER COMMAND
/×SDA
COMMAND READY TO SEND
COMMAND SENT
> PRINTON
ENTER PRINTER NAME: PRTR1
PRTR1 DEFINED AS RJE PRINTER
> COMMAND
ENTER COMMAND
/×SDA
COMMAND READY TO SEND
> RESET
ENTER RESET TYPE (CO,SU,SP,PU): CO
RESET COMPLETED
PUNCHING STARTED
PUNCHING COMPLETED
LAST CARD PUNCHED WAS CARD 2 ON RECORD
                                            34
> SPOOL
```

Figure 7. Sample \$RJE Session (Part 1 of 2)

```
ENTER SPOOL FILE NAME (NAME, VOLUME): SPOOL01, EDX002
SPOOL FILE DEFINED
> SUBMIT RJEJOB02
SUBMIT FILE READY TO SEND
FILE TRANSMISSION STARTED
FILE TRANSMISSION COMPLETED
SPOOLING STARTED
PUNCH DATA BEING RECEIVED - NO PUNCH FILE DEFINED
ENTER PUNCH FORMAT - S OR O: S
ENTER PUNCH FILE NAME (NAME, VOLUME): PCHOUTO2, EDX002
PUNCH FILE DEFINED
PUNCHING STARTED
PUNCHING COMPLETED
                                            51
LAST CARD PUNCHED WAS CARD 1 ON RECORD
> ENDSPOOL
SPOOLING COMPLETED
> COMMAND
ENTER COMMAND
/*SIGNOFF
COMMAND READY TO SEND
COMMAND SENT
$RJE2780 ENDED AT 00:00:00
> $L $PRT2780
DS1(NAME, VOLUME): SPOOLO1, EDX002
$PRT2780
              9P,00:00:00, LP= 7C00
PRINT TO $SYSPRTR? (Y OR N): N
ENTER PRINTER NAME: PRTR1
$PRT2780 ENDED AT 00:00:00
```

Figure 8. Sample \$RJE Session (Part 2 of 2)

### CHAPTER 4. HOST COMMUNICATIONS FACILITY

An application program coded in the Event Driven Language may communicate with the facilities of Installed User Program 5796-PGH, the IBM Series/1 Host Communications Facility installed on an IBM System/370 running OS/MVT or OS/VS2. The TP statement, a part of the Event Driven Language, provides you a means of performing the following general 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 may be tested by a program running on the host system.
- Test for the occurrence of an event which is set by the host system.
- Erase an event which occurred on either the Series/1 or the host system.

To configure your supervisor for the Host Communications Facility, refer to the section "System Configuration" in the <a href="System Guide">System Guide</a>.

#### Open Series/1 Data Sets

A Series/1 may only have one host data set open at a time. If a second task attempts to open a data set, it will be placed in a queue of tasks waiting to use the TP facility.

If the task currently using the TP facility attempts to open a second data set, then the currently open data set will automatically be closed and the second one will be opened.

# Host Data Set Naming Conventions

Data set names referenced by a TP instruction must consist of an alphameric character string immediately preceded by one word which specifies the length of the name field. This is most easily done by using 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). The name field must be padded on the right with blanks.

A partitioned data set and member name is specified with 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.

A data set name can be read into a text field from the console with the READTEXT instruction.

#### Host Data Set Characteristics

Host system data sets referenced in these functions must all be cataloged, single-volume, direct-access data sets, with fixed-or variable-length records. Either sequential data sets or members of partitioned data sets may be accessed. Fixed-length logical records must contain an even number of words. The data sets may be blocked or unblocked. If fixed blocked format is used, the block size must be an integral multiple of the logical record length (LRECL), not exceeding 13030.

Either sequential data sets or members of partitioned data sets may be used for the SUBMIT function. Logical records must be 80 bytes long and may be blocked or unblocked. If blocked records are used, the block size must be an integral multiple of 80.

# Host System Considerations

To ensure economical utilization of host main storage, while also providing large record capability, host main storage is shared by all Series/1 systems. The Host Communications Facility IUP region allocation determines how much buffer space is available and therefore the upper limit for host BLKSIZE. It is still possible an error code 222 (sufficient I/O buffer space unavailable) may occur because of multiple and simultaneous requests for access to data sets with very large block sizes. This is very improbable, but you are cautioned to minimize the amount of realtime during which you use the Host Communications Facility in order to minimize the probability of interference.

You are also cautioned to test for the specific error code 222 (sufficient I/O buffer space unavailable) in response to a TP OPEN and, if received, to retry your request a little later.

#### Record Sizes

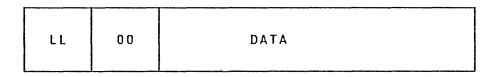
A large range of logical and physical record sizes is available to the application programmer. In selecting record size, you should understand that there is no absolute best choice. However, the following points are offered for your consideration.

 The basic disk or diskette record size on the Series/1 is 256 bytes. This is therefore 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 chosen for the TP instructions.

- A host physical record (block) size of 1536 bytes yields an efficient (80 percent) utilization of host direct access storage on an IBM 3330 disk. This also yields moderate requirements for host buffer storage.
- For unformatted data, FORTRAN IV on the host system supports either fixed-length unblocked data sets or variable-length blocked data sets.
- 4. The larger the physical record being transferred between host and Series/1 (a host logical record), the higher the effective data transfer rate which will be achieved. Also, the larger the physical record (block) being transferred between host main storage and direct access, the higher 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.
- 5. The large physical records naturally require correspondingly large buffers in your program. In order to achieve overlapped I/O, multiple buffers are required.

# 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 structure of a variable format record is shown below.



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 field shown as 00 (bytes 3 and 4 ) is reserved for use by the host system.

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

### Data Transfer Rates

The data transfer rates which may be achieved between Series/1 and the host is a function of the activity on the host and as

such will vary somewhat from time to time. Of course, the speed of transmission is also a function of 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.

#### System Status Data Set

The status functions (SET, FETCH, and RELEASE) provide a method of communication and therefore, of synchronization between programs in a distributed system environment. This function is implemented by using a shared system data set on the host computer. Programs on the host or satellite processors can communicate by writing (SET), reading (FETCH), and deleting (RELEASE) records.

In the simplest case, 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) would test for the existence of such an entry with a FETCH or RELEASE referring to the same index and key names and would receive a positive return code if the entry existed. 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. Figure 9 on page 86 graphically illustrates the previous explanation.

The FETCH, SET, and RELEASE functions can be invoked from a user-written program using the TP commands or, through the use of the Event Driven Executive \$HCFUT1 utility. The return codes that could be returned are listed in the section "Return Codes" on page 102.

PROGA	PROGRAM	A	PROGRAM A				
STATA	STATUS	PROGID, KEYA	DEFINE STATUS ID & KEY				
*							
A	TP	SET, STATA	SEND MESSAGE TO PROGB				
* A1	TP	FETCH, STATA, ERRORA	VIA HOST CHECK IF PROGB RECEIVED				
*	T.F.	reion, Stata, ERRURA	MESSAGE				
	LL THRU	IF KEY & ID STILL O					
*			., ., = -				
	COLO	A 1	CONTINUE INTERROGATION				
ERRORA	EQU	*	DELETE THE MESSAGE ON HOST				
	PROGSTO	P					
	ENDPROG						
	END						
PROGB	PROGRAM	В	PROGRAM B				
STATB	STATUS	PROGID, KEYA	DEFINE SAME STATUS ID & KEY				
*							
В	TP FETCH, STATB, ERROR = E		ERRORB FETCH MESSAGE				
*	22105 111		TED TUILS STOULLTUS BROOK				
* ME	SSAGE WA	SAGE WAS FOUND AND IS DELETED, THUS SIGNALING PROGA					
*	TP	RELEASE.STATB					
	бото	END					
ERRORB	GOTO		CONTINUE LOOKING FOR MESSAGE				
END	PROGSTOP						
	ENDPROG						
	END						

Figure 9. System Status Data Set Sample Program

The System Status Data Set has DIRECT organization. Records are written into this data set with the SET function, tested for existence with the FETCH function, or tested and deleted with RELEASE.

A STATUS entry has three possible logical parts, two of which are mandatory. These are:

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

Index entries and key fields are each 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. These are:

- 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 which has the following format:

- 1. Forward pointer a 4-byte relative record number of the next key entry or zero if this is the last one
- 2. Key name eight EBCDIC characters
- 3. 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 <u>IBM Series/1 Host Communications Facility</u> <u>Program Description and Operation Manual</u>, SH20-1819, contains more details on the use of the System Status Data Set.

#### TP Statement

The TP statement supports only the single line BSC adapter in point-to-point leased line mode. The following list shows the required TP statement, or required sequence of TP statements, to perform each of the general functions. These statements are coded in your Event Driven Language application program, which runs on the Series/1 end of the BSC link.

### Examples of Use

1. Write data from the Series/1 to a host data set.

Requires: TP OPENOUT,...

TP WRITE,...

TP CLOSE,...

2. Read data from a host data set to the Series/1.

Requires: TP OPENIN,...

TP READ,...

TP CLOSE,...

3. Submit a background job to the host system.

Requires: TP SUBMIT,...

4. Obtain the time and date from the host system.

Requires: TP TIMEDATE,...

5. Set, on the host system, the occurrence of a Series/1 event, so that it may be tested by a program running on the host system.

Requires: TP SET,...

6. Test for the occurrence of an event set by a program running on the host system.

Requires: TP FETCH,...

7. Erase the record, on the host system, of an event which was set by either the host system or the Series/1.

Requires: TP RELEASE,...

## TP Statement Syntax

Each of the forms of the TP instruction is described starting with "TP CLOSE." The use of each function is shown in "Example Transfer a Series/1 Data Set to the Host" on page 105 and "Example Transfer a Host Data Set to the Series/1" on page 106. Certain standard information is described on the following pages.

### TP CLOSE

TP CLOSE terminates a transfer operation. This instruction is used to terminate either an operation begun with TP OPENOUT,... or with TP OPENIN,....

### Notes:

- 1. If an error occurs, an open data set will be automatically closed by the system. The only time that a TP CLOSE must be issued is when a data set transfer is being terminated and no errors have occurred. For instance, this would occur if only 10 records were being written to or read from a data set capable of containing 20 records.
- 2. The return code should always be tested after issuing a TP CLOSE because some errors will only be detected at this time; for example, 50 and 51. Return codes are shown in Figure 10 on page 102, Figure 11 on page 103 and Figure 12 on page 104.
- 3. While you have an open data set, no one else will be able to use the facility. Use discretion in your operations.

### Syntax

label TP CLOSE, ERROR=

Required: CLOSE Defaults: None Indexable: None

### Operands Description

label The optional symbolic name of the TP statement.

CLOSE Coded as shown. Specifies operation termination.

ERROR= Use this operand to specify the first instruction of the routine to be invoked if an error condition occurs during the execution of this operation. If this operand is not specified, control will be returned to the next instruction after this one and you must test for errors.

# TP FETCH

TP FETCH tests for the existence of a specific record in the System Status Data Set on the host system and optionally reads in the associated data record.

### Syntax

label TP FETCH,stloc,length,ERROR=,P2=,P3=

Required: FETCH, stloc Defaults: length=0 Indexable: stloc, length

Operands Description

label The optional symbolic name of the TP statement.

FETCH Coded as shown.

stloc The label of a STATUS instruction. Refer to the <a href="Language Reference">Language Reference</a> for a description of this instruction.

instruction.

length A count specifying the length, in bytes, of the data portion of the status record to be received. A count of zero indicates that no data is to be received.

The maximum value of this field is 256.

ERROR= Use this operand to specify the first instruction of the routine to be invoked if an error condition occurs during the execution of this operation. If this operand is not specified, control will be returned to the next instruction after this one and

you must test for errors.

Pn= The optional labels to be affixed to the stloc and

length operands, respectively.

### TP OPENIN

TP OPENIN prepares to read data from a host data set.

# <u>Syntax</u>

label TP OPENIN,dsnloc,ERROR=,P2=

Required: OPENIN, dsnloc

Defaults: None Indexable: dsnloc

Operands Description

label The optional symbolic name of the TP statement.

OPENIN Coded as shown. Specifies an input operation.

dsnloc The label of a TEXT instruction which specifies the fully qualified name of a host data set of standard format as detailed in "Host Data Set Naming

Conventions" on page 82.

This may be either (1) a sequential data set or (2)

a partitioned data set with member name included.

ERROR= Use this operand to specify the first instruction

of the routine to be invoked if an error condition occurs during the execution of this operation. If this operand is not specified, control will be returned to the next instruction after this one and

you must test for errors.

P2= The optional label to be affixed to the dsnloc

operand.

### TP OPENOUT

TP OPENOUT prepares to transfer data to a host data set.

## Syntax

label TP OPENOUT, dsnloc, ERROR=, P2=

Required: OPENOUT, dsnloc

Defaults: None Indexable: dsnloc

<u>Operands</u> <u>Description</u>

The optional symbolic name of the TP statement. label

OPENOUT Coded as shown. Specifies an output operation.

dsnloc The label of a TEXT instruction which specifies the fully qualified name of a host data set of standard format as detailed in "Host Data Set Naming Conventions" on page 82.

This may be either (1) a sequential data set or (2) a partitioned data set with member name included.

ERROR= Use this operand to specify the first instruction

of the routine to be invoked if an error condition occurs during the execution of this operation. If this operand is not specified, control will be returned to the next instruction after this one and

you must test for errors.

P2= The optional label to be affixed to the dsnloc

operand.

# TP READ

TP READ receives a data record from the host system.

## Syntax

label TP READ, buffer, count, END=, ERROR=, P2=, P3=

Required: READ, buffer Defaults: count=256 Indexable: buffer, count

<u>Operands</u>	Description
label	The optional symbolic name of the TP statement.
READ	Coded as shown. Specifies that a record is being received.
buffer	The label of the data buffer into which the record is to be stored. This buffer should be generated with or conform to the specifications of a BUFFER statement specifying TPBSC.
count	The maximum number of bytes which may be transferred. For variable length records, this includes the 4-byte RDW as shown in "Variable Length Records" on page 84.
END=	Use this operand to specify the first instruction of the routine to be invoked if an "End of Data Set" condition is detected (return code 300). If this operand is not specified, an EOD will be treated as an error.
ERROR=	Use this operand to specify the first instruction of the routine to be invoked if an error condition occurs during the execution of this operation. If this operand is not specified, control will be returned to the next instruction after this one and you must test for errors.
Pn=	The optional labels to be affixed to the buffer and count operands, respectively.

## TP RELEASE

TP RELEASE deletes a specific record in the System Status Data Set on the host system and optionally reads the associated data record.

#### Syntax

TP RELEASE, stloc, length, ERROR=, P2=, P3= label

Required: RELEASE, stloc

Defaults: length=0

Indexable: stloc, length

<u>Operands</u> Description

label The optional symbolic name of the TP statement.

RELEASE Coded as shown.

The label of a STATUS instruction. Refer to the stloc Language Reference for a description of this instruction.

A count specifying the length, in bytes, of the data length portion of the status record to be received. A count of zero indicates that no data is to be transmitted.

The maximum value of this field is 256.

Use this operand to specify the first instruction ERROR= of the routine to be invoked if an error condition occurs during the execution of this operation. If this operand is not specified, control will be returned to the next instruction after this one and

you must test for errors.

The optional labels to be affixed to the stloc and Pn=

length operands, respectively.

# TP SET

TP SET writes a record in the System Status Data Set on the host system.

## Syntax

label TP SET,stloc,length,ERROR=,P2=,P3=

Required: SET, stloc Defaults: length=0

Indexable: stloc, length

<u>Operands</u>	Description
label	The optional symbolic name of the TP statement.
SET	Coded as shown.
stloc	The label of a STATUS instruction. Refer to the Language Reference for a description of this instruction.
length	A count specifying the length, in bytes, of the data portion of the status record to be transmitted. A count of zero indicates that no data is to be transmitted. The maximum value of this field is 256.
ERROR=	Use this operand to specify the first instruction of the routine to be invoked if an error condition occurs during the execution of this operation. If this operand is not specified, control will be returned to the next instruction after this one and you must test for errors.
Pn=	The optional labels to be affixed to the stloc and length operands, respectively.

## TP SUBMIT

TP SUBMIT submits a job to the host batch job stream.

### Syntax

label TP SUBMIT,dsnloc,ERROR=,P2=

Required: SUBMIT, dsnloc

Defaults: None Indexable: dsnloc

## <u>Operands</u> <u>Description</u>

label The optional symbolic name of the TP statement.

SUBMIT Coded as shown.

dsnloc The label of a TEXT instruction which specifies the name of a host data set containing the job (JCL and optional data) to be submitted.

This may be either:

- 1. TEXT "dsname" for a sequential data set, or
- TEXT "dsname(membername)" for a partitioned data set.

In systems with a HASP/Host Communications Facility interface, specifying DIRECT for dsnloc allows immediate transmission of data records to the job stream without employing an intermediate host data set. To use this facility, issue:

TP SUBMIT, DIRECT

followed by a series of

TP WRITE, buffer, 80

instructions, one for each job stream record, terminated with a

TP CLOSE

- ERROR= Use this operand to specify the first instruction of the routine to be invoked if an error condition occurs during the execution of this operation. If this operand is not specified, control will be returned to the next instruction after this one and you must test for errors.
- P2= The optional label to be affixed to the dsnloc operand.

## TP TIMEDATE

TP TIMEDATE obtains the current time of day (hours, minutes, and seconds) and the date (month, day, and year) from the host system.

#### Syntax

label TP TIMEDATE,loc,ERROR=,P2=

Required: TIMEDATE, loc

Defaults: None Indexable: loc

Operands Description

label The optional symbolic name of the TP statement.

TIMEDATE Coded as shown.

The label of the 6-word data area where time of day and date will be stored as hours, minutes, seconds,

month, day, and year.

ERROR= Use this operand to specify the first instruction of the routine to be invoked if an error condition occurs during the execution of this operation. If

this operand is not specified, control will be returned to the next instruction after this one and

you must test for errors.

P2= The optional label to be affixed to the loc operand.

# TP WRITE

TP WRITE sends a data record to the host system.

## Syntax

label TP WRITE, buffer, count, END=, ERROR=, P2=, P3=

Required: WRITE, buffer Defaults: count=256 Indexable: buffer, count

<u>Operands</u>	Description
label	The optional symbolic name of the TP statement.
WRITE	Coded as shown. Specifies that a record is being sent.
buffer	The label of the data buffer which contains the record to be transmitted. This buffer should be generated with, or conform to the specifications of, a BUFFER statement specifying TPBSC.
count	The number of Series/1 bytes to be transferred. For variable length records, this includes the 4-byte RDW as shown in "Variable Length Records" on page 84.
END=	Use this operand to specify the first instruction of the routine to be invoked if an "End of Data Set" condition is detected (return code 400). If this operand is not specified, an EOD will be treated as an error.
ERROR=	Use this operand to specify the first instruction of the routine to be invoked if an error condition occurs during the execution of this operation. If this operand is not specified, control will be returned to the next instruction after this one and you must test for errors.
P n =	The optional labels to be affixed to the buffer and count operands, respectively.

## Return Codes

Program execution will be halted until the operation is complete, and the first word of the TCB (taskname) must be tested to determine if the operation was successful. The return codes are shown in Figure 10, Figure 11 on page 103 and Figure 12 on page 104.

Note: If an error is detected, an open data set is automatically closed for you.

Code	Description	Module
-1	Successful completion	Supervisor
1	Illegal command sequence	Supervisor
2	TP I/O error	Supervisor
3	TP I/O error on host	HCFCOMM
4	Looping bidding for the line	Supervisor
5	Host acknowledgement to request code was neither ACKO, ACK1, WACK, or a NACK	Supervisor
6	Retry count exhausted - last error was a timeout; the host must be down	Supervisor
7	Looping while reading data from the host	Supervisor
8	The host responded with other than an 'EOT' or an 'ENQ' when an 'EOT' was expected	Supervisor
9	Retry count exhausted - last error was a "modem interface check"	Supervisor
10	Retry count exhausted - last error was not a timeout, modem check, block check or overrun	Supervisor
11	Retry count exhausted - last error was a transmit overrun	Supervisor
50	I/O error from last I/O in DSWRITE	DSCLOSE
51	I/O error when writing the last buffer	DSCLOSE
100	Length of DSNAME is zero	HCFCOMM
101	Length of DSNAME exceeds 52	HCFCOMM
102	Invalid length specified for I/O	HCFINIT

Figure 10. TP Return Codes (Part 1 of 3)

C = d =	Donation	Module
Code	Description	Module
200	Data set not on volume specified for	HCFINIT
	controller	
201	Invalid member name specification	DSOPEN
202	Data set in use by another job	DSOPEN
203	Data set already allocated to this task	DSOPEN
204	Data set is not cataloged	DSOPEN
205	Data set resides on multiple volumes	DSOPEN
206	Data set is not on a direct access	
	device	DSOPEN
207	Volume not mounted (archived)	DSOPEN
208	Device not online	DSOPEN
209	Data set does not exist	DSOPEN
211	Record format is not supported	DSOPEN
212	Invalid logical record length	DSOPEN
213	Invalid block size	DSOPEN
214	Data set has no extents	DSOPEN
216	Data set organization is partitioned and	
	no member name was specified	DSOPEN
217	Data set organization is sequential and	
	a member name was specified	DSOPEN
218	Error during OS/ OPEN	DSOPEN
219	The specified member was not found	DSOPEN
220	An I/O error occurred during a	
	directory search	DSOPEN
221	Invalid data set organization	DSOPEN
222	Sufficient I/O buffer space unavailable	DSOPEN
300	End of an input data set	DSREAD
301	I/O error during an OS/ READ	DSREAD
302	Input data set is not open	DSREAD
303	A previous error has occurred	DSREAD

Figure 11. TP Return Codes (Part 2 of 3)

Code	Description	Module
400	End of an output data set	DSWRITE
401	I/O error during an OS/ WRITE	DSWRITE
402	Output data set is not open	DSWRITE
403	A previous error has occurred	DSWRITE
404	Partitioned data set is full	DSCLOSE
700	Index, key, and status record added	SET
701	Index exists, key and status added	SET
702	Index and key exist, status replaced	SET
703	Error - Index full	SET
704	Error - Data set full	SET
710	I/O Error	SET
800	Index and key exist	FETCH
801	Index does not exist	FETCH
802	Key does not exist	FETCH
810	I/O error	FETCH
900	Index and/or key released	RELEASE
901	Index does not exist	RELEASE
902	Key does not exist	RELEASE
910	I/O error	RELEASE
1×××	An error occurred in a subordinate module during SUBMIT. 'xxx' is the code returned by that module.	S7SUBMIT

Figure 12. TP Return Codes (Part 3 of 3)

### Example Transfer a Series/1 Data Set to the Host

In the following example, a Series/1 data set, which is entered by the user at program load time, is written to a 256-byte data set on the host. The user will be prompted for a target host data set.

```
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
               DS1, BUFFER, ERROR = ERR2, END = TPCLOSE
DSREAD
         READ
                WRITE A RECORD TO HOST
TPWRITE
         TP
                WRITE, BUFFER, 256
         ΙF
                (WRITASK, EQ, -1), GOTO, DSREAD
                                               ..OK?
ERR1
         MOVE
               SWITCH, 1
                                           ..WRITE ERROR
         GOTO
               TPCLOSE
ERR2
         MOVE
               SWITCH, 2
                                          ..READ ERROR
       LOSE DATA SET AND PRINT MESSAGE AS APPROPRIATE
TPCLOSE
         ΤP
               CLOSE
ERRSW
         GOTO
               (RETO, RET1, RET2, RET3), SWITCH
RETO
         PRINTEXT '****READ/WRITE SUCCESSFUL*****a'
         PROGSTOP
RET1
         PRINTEXT '****WRITE UNSUCCESSFUL*****a'
         PROGSTOP
         PRINTEXT '****READ UNSUCCESSFUL*****
RET2
         PROGSTOP
         PRINTEXT '****TP OPEN UNSUCCESSFUL*****a'
RET 3
         PROGSTOP
SWITCH
         DATA F'0'
DSNAME
         TEXT
               LENGTH=40
BUFFER
         BUFFER 256, TPBSC
         ENDPROG
         END
```

### Example Transfer a Host Data Set to the Series/1

In the following example, a host data set which is entered by the user at the prompt "HOST DATASET: ", is read into a preallocated data set on a Series/1 volume. At program load time the user is prompted for the target Series/1 data set.

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

## \$HCFUT1 Utility Program

\$HCFUT1 is a utility program that uses the Host Communications Facility on the Series/1 to interact with the Host Communications Facility on the System/370. \$HCFUT1 contains four host-related data set functions. These are:

- Read a data set from the host.
- Write a data set to the host.
- Submit a job to the host.
- Status Set, Fetch, and Release records in the System Status Data Set.

The table below lists the commands and their codes:

١			
I	?	Help	
ı	END	End	
l	FEtch	Fetch status	
I	RELease	Release status	
1	READDATA	Read host	
1	READ80	Read 80-byte records and write two 80-byte	
		records in one disk sector	
1	READOBJ	Read 80-byte records and write three 80-byte	
١		records in one disk sector	
	SEt	Set status	
	SUbmit	Submit a job	
	WRite	Write to host	
L			

### Notes:

- See "Host Data Set Naming Conventions" on page 82 and "Host Data Set Characteristics" on page 83.
- See "System Status Data Set" on page 85. Appendix B of the IBM Series/1 Host Communications Facility Program Description and Operation Manual, SH20-1819, contains more details on its use.
- The Host Communications Facility IUP, program number 5796-PGH, is required on the host System/370.
- Host Communications Facility must be installed and configured on the Series/1.

## READDATA

READDATA transfers a data set from the host to the Series/1. The host logical record size is assumed to be 256 bytes.

There are three items of control information to be specified at the time of execution. These items are:

DS1

The 1-8 character name of the Series/1 data set to which data is to be transferred, and its volume name, if not the IPL volume.

Record Count

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

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

PGM(NAME, VOLUME): \$HCFUT1

DS1(NAME, VOLUME): DATAFIL2, EDX001 \$HCFUT1 8P,08.15.30, LP=4B00

COMMAND (?): READDATA

NO. OF RECORDS TO READ(0=ALL): 0

DSNAME: \$1.EDX.TESTIN.DATA

END AFTER 40

COMMAND (?):

## READ80 and READOBJ

READ80 and READ0BJ 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 \$LINK 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 \$LINK or \$UPDATE.

Both READ80 and READOBJ are invoked in a manner similar to "READDATA" on page 108.

## SET, FETCH, and RELEASE

The status commands are used to perform, from a terminal, any of the three functions, SET, FETCH, and RELEASE, on the System Status Data Set. See "System Status Data Set" on page 85 and Figure 11 on page 103 for STATUS return codes.

The following is an example of the use of the SET function of \$HCFUT1. STATUS return code 700 indicates that the index, key, and status record have been added.

COMMAND (?): SE
INDEX = TESTSET
KEY = NEWRECD
STATUS = 700
COMMAND (?):

The following are examples of the use of the FETCH and RELEASE functions. The FETCH return code of 802 indicates that that particular key does not exist. The RELEASE return code of 900 indicates a successful release.

COMMAND (?): FE
INDEX = TESTSET
KEY = MISSING1
STATUS = 802
COMMAND (?): REL
INDEX = TESTSET
STATUS = 900
COMMAND (?):

## SUBMIT

SUBMIT causes a job to be submitted to the host job stream. See "Host Data Set Naming Conventions" on page 82 and "Host Data Set Characteristics" on page 83.

The name of the host data set containing the job control language to be submitted is specified on the Series/1 terminal. The following is a sample of the terminal printout illustrating the use of SUBMIT to submit the data set "S1.EDX.TESTSUB.CNTL".

COMMAND (?): SU

DSNAME: S1.EDX.TESTSUB.CNTL

JOB SUBMITTED ANOTHER JOB? N

COMMAND (?):

### WRITE

WRITE transfers a data set from the Series/1 to the host processor. Host data set naming conventions and characteristics are described in this chapter. The host logical record size is assumed to be 256 bytes.

There are three items of control information to be specified at the time of execution. These items are:

DS1

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

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

The name of the host data set to which the data is to be transferred. The name will consist of up to 44 characters or, 54 characters if a member of a partitioned data set.

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

DS1(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** 

COMMAND (?):

#### INTRODUCTION

The Series/1 Event Driven Executive Multiple Terminal Manager is a program which provides support, via high-level functions, for transaction-oriented applications on a Series/1. In addition, it provides the management of multiple terminals as needed to support these transactions and their various application programs. The user creates programs which interface with the Multiple Terminal Manager via CALL statements. The components of the Multiple Terminal Manager are the following:

- A program/storage manager which controls the execution and flow of the application programs within a single program area.
- A terminal/screen manager which controls the presentation of screens and communications between terminals and application programs.
- A file handling mechanism which simplifies the storage and retrieval of data on direct access devices.

Note: The reader should be familiar with the terminology used in the discussion of the TERMINAL statement in the section "System Configuration" of the <u>System Guide</u>. The syntax of the CALL statements in this chapter can be found in the <u>Language Reference</u>.

#### HARDWARE REQUIREMENTS

The minimum hardware configuration required for the Multiple Terminal Manager is as follows:

- Series/1 processor (either 4952 or 4955) with 96KB storage
- Disk storage device (either 4962 or 4963)
- An Event Driven Executive \$SYSPRTR device
- 4978/4979/3101 or ASCII terminal

A separate \$SYSLOG device is also required for receiving system messages; this device should not be included in the Multiple Terminal Manager environment in that system messages may not be displayed.

Additional hardware that may be attached to the system:

- 4978, 4979, or 3101 Models 1 or 2 terminal devices
- ASCII terminals connected via: #7850 Teletypewriter Adapter, er, #1610 controller, #2091 controller with #2092 adapter, or #2095 controller with #2096 adapter.
- 4973 or 4974 printers
- Additional direct access devices (disk or diskette)
- Additional storage

# SOFTWARE REQUIREMENTS

The minimum software requirements for executing the Multiple Terminal Manager is the Event Driven Executive V1.1. Additionally, the Event Driven Executive utilities and program preparation facilities are required for program preparation and installation of Multiple Terminal Manager applications. The following is a list of the additional software supported by the Multiple Terminal Manager:

- Indexed Access Method
- COBOL
- FORTRAN
- PL/I

## PROGRAM OPERATION OVERVIEW

The Multiple Terminal Manager is a transaction processing subsystem which executes as an application program within the Event Driven Executive system. Multiple Terminal Manager transactions are initiated by a terminal operator via a transaction selection menu (also referred to as a program selection menu). Transactions can consist of single or multiple operator prompts, and responses are processed by user applications prepared explicitly for the Multiple Terminal Manager.

Multiple Terminal Manager applications are processed in a message in/message out fashion and are automatically connected to a terminal when a transaction begins. The Multiple Terminal Manager, in turn automatically processes terminal I/O for Multiple Terminal Manager applications. Multiple Terminal Manager applications execute within the program area managed by the Multiple Terminal Manager. The applications are provided program, terminal, screen and file management services via the Multiple Terminal Manager.

### Program Management

The program management facilities allow applications to manage programs while these programs perform their respective transactional processes within a single overlay area. Because all of the Multiple Terminal Manager application programs operate in the same area, the Multiple Terminal Manager program management facilities contain the support needed to allow multiplex operation and sharing of the program area. The application programs interface with these facilities using the callable functions described in the following sections.

The program management callable functions are:

LINK: Load and Execute Program

The LINK function allows an application program to complete its own execution by loading and executing some other application program.

LINKON: Fetch Response and Execute Program

The LINKON function is a combination of the functions provided by the ACTION and LINK functions; that is, it requests an operator action and, when this action is complete, loads and executes some other application program.

#### CYCLE: Suspend Current Terminal Application

The CYCLE function allows an application program to suspend its execution to allow other applications/terminals to become active.

MENU: Return to Multiple Terminal Manager Control

The MENU function allows the application program to abort its own operation and return control to the Multiple Terminal Manager base program. The operator selection menu is then displayed on the terminal.

The application programs using these program/storage management facilities will always have the following four items associated with them:

Application Program: This is the user-written code that performs the transaction processing as required by the user. It resides in the PRGRMS volume and is loaded into the in-storage program area by the manager.

Swap Out Data Set: Resides on MTMSTORE, MTMSTR. This data set is used by the manager to save programs and data across calls to ACTION, LINK, LINKON, CYCLE, and WRITE.

Input Buffer: This buffer contains either the data last entered by the operator when the current part of the application program was entered or, the protected characters of the screen display that the application program is preparing for the next dialogue with the operator. This buffer is allocated by the Multiple Terminal Manager and is normally 2048 bytes in length.

Output Buffer: This buffer contains the unprotected characters of the screen display that the current application program is preparing for the next dialogue with the operator. These unprotected characters can either be default values, or values supplied by the application program. This buffer is allocated by the Multiple Terminal Manager and is 1024 bytes in length.

### Terminal/Screen Management

The terminal/screen management facilities provide you with a simplified method of performing the terminal handling functions that your application program may require. These facilities are described as follows:

ACTION: Fetch Operator Response

The ACTION function allows the application program to display a screen on the terminal and then obtain operator input from that display.

SETPAN: Retrieve a Screen Image from the SCRNS Volume

The SETPAN function allows the application program to request a specified screen be retrieved from the SCRNS volume and loaded into the Input and Output Buffers.

SETCUR: Move Cursor to Specified Position

The SETCUR function allows the application program to reset the character position at which the terminal/screen manager will display the cursor when the screen is displayed.

BEEP: Set Audible Alarm

The BEEP function allows the application program to activate the audible alarm, if this feature is supported by the terminal, on the next output as a signal to the terminal operator.

CHGPAN: Change Panel

The CHGPAN function is used to notify the terminal manager of changes to the number of protected/unprotected characters of a screen in the input buffer. As a result of this function, the terminal manager will know how many unprotected data characters to write on the next output operation. This function allows an application program to dynamically modify or create a screen image.

FTAB: Describe Unprotected Input Fields

The FTAB function is used to set up a table that describes the unprotected input fields placed in the Input Buffer after a SETPAN or CHGPAN is issued. This function is useful in cursor positioning.

### WRITE: Output to an ASCII Terminal

This function is provided for those applications which utilize ASCII terminals such as the Teletype\* ASR 33/35. This function executes similar to the functions described in the section "Program Management" on page 115, in that the application program does not remain in storage while the buffer is being written; hence, the manager returns control to the calling application program at the next sequential instruction.

\* Trademark of the Teletype Corporation

## File Management

The file management facilities of the Multiple Terminal Manager provide common, easy-to-use support for all disk data-transfer operations as needed for the transaction-oriented application programs. These facilities provide support for both indexed and direct files under the control of a single callable function. The file management facilities consist of the FILEIO function.

#### FILEIO: Perform Disk I/O

This function allows the application program to perform read and write operations to disk using either indexed or direct accessing.

# Multiple Terminal Manager Operation

The Multiple Terminal Manager is invoked using the Event Driven Executive \$L command (\$L \$MTM,PRGRMS). When this command is issued, the Multiple Terminal Manager program manager is loaded into storage and activated. The first program activated by the program manager is the Multiple Terminal Manager initialization program.

## Multiple Terminal Manager Initialization Program

This program determines the number of terminals that are being controlled and prepares the tables and in-storage control blocks necessary to support those terminals. The initialization program LOADs and initializes a terminal server for each terminal that is to be controlled by the Multiple Terminal Manager. When initialization is complete, control is returned to the program manager.

### Terminal Server Programs

The terminal server programs perform all input/output and interrupt handling functions for those terminal devices operating under the control of the Multiple Terminal Manager. There is one terminal server program for each terminal assigned to the Multiple Terminal Manager.

## Application Program Manager

The application program manager controls the contents of the program area and the execution of programs within that area.

#### Multiple Terminal Manager Utilities

The utility program support provided with the Multiple Terminal Manager consists of operator service functions which assist you in the operation of your Multiple Terminal Manager system. These utilities are described as follows:

**Terminal Connection Facilities:** The Multiple Terminal Manager supervisor program provides the operator with the facilities to disconnect and reconnect terminals during the normal Multiple Terminal Manager operation. These services are performed by the following operator commands:

### **DISCONNECT:** Turn Off Specified Terminals

This facility allows the operator to shut down all or individually-specified terminals on the Multiple Terminal Manager system. If the operator requests a terminal, which is currently involved in a transaction, to be disconnected, that terminal will be allowed to complete its associated transaction before being disconnected.

#### **RECONNECT:** Turn On Specified Terminals

This facility allows the operator to restore a disconnected terminal (via DISCONNECT) back into operation.

**Terminal Activity Report:** This report utility allows the operator to display the names and current status of the terminals under control of the Multiple Terminal Manager.

**Programs Report:** This report utility allows the operator to display the names and sizes of Multiple Terminal Manager application programs.

Screens Report: This report utility allows the operator to display screen formats developed for Multiple Terminal Manager applications.

### Sign-On/Sign-Off

The Multiple Terminal Manager provides an optional facility to support operator sign-on and user provided sign-off. This support is provided when the Multiple Terminal Manager user wishes to restrict the use of the Multiple Terminal Manager system to only user-specified authorized personnel.

#### Data Files

The Multiple Terminal Manager maintains several files on disk to assist in the operation of the program and its users. The following is a list of these data files:

- SCRNS Volume This volume contains the formatted screen displays which are built by the Event Driven Executive \$IMAGE utility.
- TERMINAL File This file describes the terminals that are to be controlled by the Multiple Terminal Manager.
- PRGRMS Volume This volume contains the Multiple Terminal Manager and user application programs.
- MTMSTORE File This file is used by the program manager as a work file primarily for saving and restoring programs across calls to the Multiple Terminal Manager.

#### APPLICATION PROGRAM INTERFACE

The Multiple Terminal Manager provides the Series/1 Event Driven Executive user with a set of high-level functions designed to simplify the definition of "transaction oriented" applications, such as inquiry, file update, data collection, and order entry.

"Transaction oriented" means that program execution is driven by operator actions, typically, responses to prompts from the system. For example, a program executing under control of the Multiple Terminal Manager displays a "menu" screen offering the operator a choice of functions. Based on the operator's selection, the application program then performs processing operations, such as reading information from a data file, displaying the data at the terminal, and waiting for the next response.

This "prompt-response-process" cycle between the Series/1 program and the terminal operator is the basic principle for the design of applications using the Multiple Terminal Manager.

The terminal manager simplifies such transactions by:

- Automatically allocating input and output buffers for the application program.
- Performing I/O operations to access fixed screen formats from the screen file. The term "screen" in this discussion refers to the image which is displayed on the screen of an IBM 4979, 4978, or 3101 (in block mode) terminal. Fixed screen formats consist of protected data and definitions of possible areas for data input. On other systems, these are referred to as "Maps", "Formats", or "Panels". Screens are built via the Event Driven Executive \$IMAGE utility.
- Returning control to the user program to allow modification of the buffers containing the screen (if desired).
- Performing the set of I/O operations involved in writing the screen to the terminal, filling in unprotected fields with user-defined output data, and reading the data entered by the operator before returning control to the application program that requested the action. (The terminal manager assumes that each ACTION request involves both output and input operations, thus eliminating the need for the application program to make separate requests).

In addition, the Multiple Terminal Manager provides storage, file, and program management services, terminal transaction statistics, and sign on/off facilities for password validation. Error recovery for I/O and program check conditions are provided by the Event Driven Executive.

Series/1 Multiple Terminal Manager applications can be written in EDL, assembler language, COBOL, FORTRAN IV, or PL/I. Disk I/O can be performed by an application program using indexed or direct access methods. Terminal support is provided for locally attached IBM 4979, 4978, and 3101 display terminals and ASCII compatible terminals attached via the #7850, #1610, #2091 with #2092, or #2095 with #2096 adapters. See Figure 1 on page 6 for a description of devices and attachments.

## | Considerations for the IBM 3101 Model 2 Terminal

The Multiple Terminal Manager supports the IBM 3101 Model 2 terminal in full screen mode ("block mode"). This support is only for Multiple Terminal Manager based application programs; other applications are not supported. In particular, screen design using the Event Driven Executive \$IMAGE utility must be performed on a 4978 or 4979. Throughout this chapter, any discussion of the 3101 refers to the Model 2 operating in block mode unless specified otherwise.

3101 support performs a subset of the functions equivalent to the support for IBM 4978 and 4979 terminals. That is, from the programming perspective, the 3101, 4978, and 4979 terminals are functionally very similar. However, they are operationally different in that the 3101 uses "attribute characters" to define fields. Multiple Terminal Manager support for the 3101 places an attribute character just prior to and following each input field, and at the first position on the screen.

Attribute characters appear as protected blanks on the display screen. Hence, the characters preceding and following an input field shall each appear as a protected blank. The same is true of the first character on the screen. These attribute characters should be taken into account and allowed for when designing screen images.

The maximum number of unprotected fields that can be displayed is 127.

Any invalid (unprintable) characters encountered by the 3101 will cause the alarm to ring. This condition might occur, for instance, when displaying a non-EBCDIC disk or diskette record. The Multiple Terminal Manager will convert to blanks, any nulls (X'00') found in an unprotected data stream to help avoid this condition.

The keys on the 3101 are labelled differently than the 4978 and 4979. The SEND key performs the same function as the ENTER key. Furthermore, the Program Function keys on the 3101 require that the ALT key on the lower right hand side of the keyboard be pressed as well as the appropriate numeric key. The PF6 key when pressed (hardcopy screen print) however, will not cause the screen image to be printed.

## Multiple Terminal Manager Components

Major components of the Multiple Terminal Manager for the application programmer are:

- Functions (callable routines)
- User application programs
- TERMINAL file
- Screen formats

The functions provided by the Multiple Terminal Manager are callable routines that perform terminal, disk and diskette input/output operations and, control the execution of application programs. Program execution and terminal I/O are combined in most instances; for example, the LINK function causes a new program to be loaded and executed. If the current screen format has not yet been displayed, LINK also causes the screen to be written to the terminal.

The program-execution control and terminal I/O functions include:

- A routine (ACTION) to initiate the "prompt-response" terminal I/O operation
- Two routines (LINK and LINKON) to link to a new program from the currently executing program
- A routine (MENU) to terminate program execution and return control to the Multiple Terminal Manager
- A routine (CYCLE) to voluntarily give up control of the program area to other users. This allows a user-controlled form of time sharing.

In addition, the following functions are used with 4978, 4979, or 3101 terminals. These routines can be executed prior to a CALL ACTION to initiate a terminal I/O cycle:

 A routine (SETPAN) to retrieve a screen into the input and output buffers

123

- A routine (SETCUR) to override the initial cursor position defined for that screen format
- A routine (BEEP) to request the audible alarm (if available) be sounded on the next terminal I/O cycle
- A routine (CHGPAN) to notify the terminal manager of changes to a screen before it is written
- A routine (FTAB) to build a table which describes the position and length of unprotected fields in the Input Buffer.

For the ASCII terminals, the following functions are provided:

- A routine (ACTION) to write to the terminal and read a reply.
- A routine (WRITE) to write to the terminal without waiting for an operator response. Multiple writes may be used to write lengthy messages, with the last message being written via ACTION.
- A routine (BEEP) to cause a bell character to be included in the next output line.

The disk I/O function provides the following for disk and diskette files:

- Automatic open of the requested file
- Indexed Access Method file support
- Direct file support
- Storage conservation through automatic open and close functions

User application programs can be executed by the operator via a selection from the primary menu or by a program via a call to LINK or LINKON. A primary menu is used only for program selection. The application programmer/terminal operator need only specify the program name. The Multiple Terminal Manager performs the operations necessary to load the program and control its execution. User programs reside in the volume PRGRMS.

The TERMINAL file is another basic element that describes the terminals to run under the terminal manager. In this file, the user specifies the terminal type, the name of the terminal, the screen to be used as the primary menu screen, and whether or not sign-on is required. The TERMINAL file provides flexibility to the user; that is, terminals can be added or deleted without rebuilding the terminal manager. The TERMINAL file resides in the volume PRGRMS.

Screen formats are used by application programs and the Multiple Terminal Manager itself. Each screen is a data set in the volume SCRNS and defines protected fields and default unprotected fields. The following screens are predefined in the SCRNS volume:

IPLSCRN The initial program load (IPL) screen that is displayed when the Multiple Terminal Manager task set starts.

SCRNSREP Used by the Screens Report Utility

SIGNONSC The sign-on screen (displayed if a sign-on procedure is specified for the terminal).

MENUSCRN A sample primary menu screen for program selection; however, the user can select any screen as a menu screen.

These screens are provided as samples and can be modified to suit individual requirements. You can define additional screens by using the Event Driven Executive \$IMAGE utility.

The following are examples of the predefined screens in the SCRNS volume.

### IPLSCRN

* * <b>*</b>	**********************	* <b>*</b>
×	IPLSCRN	×
*	EVENT DRIVEN EXECUTIVE	×
*	MULTIPLE TERMINAL MANAGER	×
×		×
×		×
*		*
×		×
×	HIT ENTER OR A FUNCTION KEY TO START THE MULTIPLE	×
×	TERMINAL MANAGER FOR THIS TERMINAL.	×
×		×
×	5719-MS1 COPYRIGHT IBM CORP 1979	×
* * *	* * * * * * * * * * * * * * * * * * * *	¥¥

The next example shows the sign-on screen.

#### SIGNONSC

******	<b>****</b> *****	****	** <b>*</b> *	* <b>*</b> *	<del>***</del>	* <b>*</b> *	* <b>* *</b> *	* * * <del>*</del>	* <del>*</del> *	* * * *	* <b>*</b> * *	****
×										SI	GNON	<b>*</b>
×	EVENT	DRIV	EN EX	ECL	JTIVI	Ε						×
*	MULTIP	LE TE	RMINA	LN	1414	SER						*
×												*
<b>*</b> \$\$\$\$\$\$	IIIIIIII	GGG	GGG	N		Ν	00	00	Ν		N	×
*S S	I	G	G	11	1	Ν	0	0	N	1	N	×
* SSS	I	G		N	N	Ν	0	0	N	N	N	×
<b>*</b> \$\$\$\$	I	G	GG	Ν	N	N	0	0	Ν	N	Ν	×
*S S	I	G	G	N	N	N	0	0	N	N	N	×
<b>*</b> \$\$\$\$\$\$	IIIIIIII	GGG	GGG	Ν		Ν	00	00	Ν		NN	×
*												×
*												*
* #	????????	PA	SSWOR	D =	==> '	???	?					×
*												×
<b>*</b>												×
******	·****	****	****	* * <b>*</b>	(***	* <del>*</del> *	***	* * * <b>*</b>	<b>* *</b>	<b>*</b> * *	* <del>* * </del>	****

This last example is the MENUSCRN.

Errors encountered by the Multiple Terminal Manager in the primary menu mode are written protected at the first 20 character positions of a screen. User-written primary menus (defined by the TERMINAL data set) should be designed with this taken into account.

The Multiple Terminal Manager responds to an interrupt from a terminal by loading the requested program specified by program name or program function key selection. The terminal manager routes subsequent operator entries to the associated program. Two program function keys are reserved:

- PF3 signals the Multiple Terminal Manager to terminate the current program and display the menu screen.
- PF6 signals Event Driven Executive to print the contents of the current screen on the device specified by the HDCOPY parameter of the TERMINAL statement for 4978/4979 terminals only. Normally, this device is the device specified for \$SYSPRTR.

## **Program Execution**

The Multiple Terminal Manager uses a single-thread approach to program execution, that is, only one application is resident at one time.

When a program is initially requested for execution (terminal operator selects by name or PF key), a copy of the program is loaded into the terminal manager program area.

When the program requests an operator response, the program is swapped out to disk and other terminals may use the program area while the operator is keying in new data. When the response is completed and the program area is available, the program is read into the program area from the swapped out data set and the program is given control at the next sequential instruction after the instruction that caused the swap out. The swap data set is MTMSTORE residing on the volume MTMSTR.

## User Program Organization

All programs must be written to operate in a conversational mode. That is, each program (or linked sequence of programs) is expected to receive data from a terminal and then send data back to the same terminal.

Upon initiation, each user program automatically receives a list of parameters. The parameters are:

## Input Buffer Address

This is the address of a buffer used for two distinct purposes: to contain the protected data defining a screen format before an ACTION and, to contain the data input from the terminal after an ACTION. After a call to SETPAN , the Input Buffer contains a 24 X 80 (1920) byte image of the screen, where unprotected fields are defined by strings of null characters (zeroes). A call to ACTION writes the screen image from the Input Buffer to the terminal. After the operator presses ENTER or a PF key, ACTION reads the data found in the unprotected fields into the Input Buffer. The input data fields are contiguous and start at the beginning of the buffer. Input from ASCII terminals (such as teletypewriters) is read from the device with the change-of-direction character removed and backspace characters converted to a logical backspace in the Input Buffer (that is, backspace characters and a corresponding number of characters preceding them are not in the buffer). This buffer is 2048 bytes in length; however, only the first 1920 bytes are used for protected output. The remainder of the buffer contains unusable information and is to be ignored.

<u>Note</u>: The output function described above is also performed by CYCLE, LINK, and LINKON; of these, only LINKON also performs the input function.

Initially, this buffer contains the characters entered on the terminal's menu screen for the first entry to a program. The name of the program must be the first eight characters. Additional characters are not used by the manager but are passed to the program. These extra characters can be used for programs which minimize operator interaction by allowing the operator to enter a complete request on the menu screen and thus avoid the need for intermediate menus or prompts.

#### Output Buffer Address

This is the address of a buffer which is also used for two purposes. It contains "default data" to be written by ACTION into the unprotected portions of the screen. That is, a call to SETPAN reads concatenated data defined by \$IMAGE into the Output Buffer. A subsequent call to ACTION writes the data from the buffer to the unprotected fields. If more characters are in the Output Buffer than there are unprotected positions on the screen, the excess characters are lost. The Output Buffer is set to blanks after a return from CALL ACTION.

The Output Buffer is also used for passing data between programs, when one LINKs to another. Prior to a LINK to another program, a program may store data in the Output Buffer. The second program will find that data in its Output Buffer.

## Terminal Environment Block (TEB)

This is the address of a control block which contains information about the terminal that initiated this program.

## <u>Interrupt Information Byte (IIB)</u>

This is the address of a word (16 bits) in storage containing, in the low-order half of the word, a code indicating the status of the prior I/O to or from this terminal.

For a 4978/4979/3101, this is always the numeric value representing the interrupting key which was pressed as part of an operator response. Since there is no WRITE available to 4978/4979/3101 this code never reflects the status of an output operation.

For ASCII terminals, this value is the return code from a READTEXT operation issued by the Multiple Terminal Manager.

The following figure provides a programmer's view of the contents of the Input and Output Buffers at various stages in the terminal manager operation cycle.

BUFFER CONTENTS UPON ENTRY TO APPL PROGRAM	INPUT BUFFER	OUTPUT BUFFER
FROM CALL ACTION	UNPROTECTED DATA READ FROM SCREEN	BLANKS (X'40')
FROM CALL LINK	BLANKS (X'40')	UNCHANGED FROM CALLING PGM
FROM CALL LINKON	UPROTECTED DATA READ FROM SCREEN	BLANKS (X'40')
FROM CALL CYCLE	BLANKS (X'40')	UNCHANGED
FROM CALL SETPAN	PROTECTED DATA FROM NEW SCREEN PANEL	UNPROTECTED DATA FROM NEW SCREEN PANEL

ACTION TAKEN UPON BUFFER CONTENTS BY FUNCTION CALL	INPUT BUFFER	OUTPUT BUFFER
BY CALL ACTION	WRITTEN PROTECTED IF CALL SETPAN HAD BEEN ISSUED	WRITTEN INTO UNPROTECTED FIELDS ON SCREEN
BY CALL LINKON	SAME AS BY CALL ACTION	SAME AS BY CALL ACTION
BY CALL LINK	SAME AS BY CALL ACTION	SAVED
BY CALL CYCLE	SAME AS BY CALL ACTION	SAME AS BY CALL LINK

Program Contents During 4978/4979/3101 Buffer Operation

# Controlling the Logic Flow of Programs

**Program Calling Parameters:** Application programs use the EDL parameter passing facilities for passing parameters to the Multiple Terminal Manager.

## For example:

SCRNX DC CL8'SCRN10' SCREEN PANEL NAME RC DC F'0' RETURN CODE FIELD

This example passes the addresses of the screen name and return code field to the Multiple Terminal Manager screen manager.

Five callable functions are provided to control I/O to terminals and to control the execution of user programs. They are ACTION, LINK, LINKON, WRITE, and CYCLE.

ACTION and WRITE perform terminal I/O. LINK and LINKON control the loading of user programs to service the current or the next operator input, respectively. CYCLE provides a method of time sharing the program area.

#### CALL ACTION

CALL ACTION, (buffer), (length), (crlf)

All parameters for all languages are one 16-bit word in length, unless otherwise specified as character strings.

## ACTION parameters:

buffer A buffer of EBCDIC text of any length.

length The number of characters in the buffer.

crlf A binary value of 1 specifies that the terminal is to be issued a carriage return and line feed (CRLF) after the message is sent. Any other value results in no CRLF being sent.

For ASCII terminals this routine:

- 1. Writes the specified buffer contents to the terminal
- 2. Waits for the operator to respond
- Reenters the current program at its next sequential instruction after the CALL ACTION

CALL ACTION

The Input Buffer is written protected to the screen if a CALL SETPAN or CALL CHGPAN command was executed previously during this transaction. The Output Buffer is written into the unprotected fields on the screen. The terminal then waits for operator input and reenters the current program (with operator input in the Input Buffer) at the next sequential instruction after CALL ACTION. (For IBM 4978/4979/3101 displays, a parameter list is ignored if specified.)

### CALL LINK

# CALL LINK, (pgmname)

LINK causes the named program to be loaded and executed (replacing the current program).

During the link, IBM 4978/4979/3101 terminals for which a SETPAN or CHGPAN has been issued will have the Input Buffer displayed. The Output Buffer is passed unchanged to the next program.

The program being linked to receives the standard parameter list for application programs (Input Buffer, Output Buffer, TEB, IIB).

LINK parameters:

pgmname An 8-byte (right padded with blanks, if necessary) program name.

If the program name is invalid, control returns to the next sequential instruction in this program; therefore, any return to the user from CALL LINK is an error condition.

### CALL LINKON

## CALL LINKON, (pgmname)

LINKON provides a combined ACTION and LINK function. When the operator has entered the requested information, the named program is entered at its entry point with the Input Buffer containing the unprotected characters from the screen or all entered characters from an ASCII terminal.

## LINKON parameters:

pgmname An 8-byte (right padded with blanks, if necessary) program name.

### CALL CYCLE

#### CALL CYCLE

When CALL CYCLE executes, the program may be swapped out as all other applications are given an opportunity to process inputs. The Output Buffer is preserved and the contents of the Input Buffer are lost (set to blanks). If a SETPAN or CHGPAN has been executed, the screen in the Input Buffer is displayed protected at this time to free up the Input Buffer.

After all other terminals have processed their inputs, the program is swapped into the program area and control is returned to the next sequential instruction after the CALL CYCLE.

# Communicating with ASCII Terminals

The Multiple Terminal Manager provides CALL WRITE to satisfy operator interaction to ASCII terminals for multiple output messages.

# CALL WRITE

CALL WRITE, (buffer), (length), (crlf)

CALL WRITE is for ASCII terminals only. It writes the specified buffer contents to the current terminal. While writing, other terminals are permitted to operate. When I/O is complete, the current user program is reloaded and reentered at the next sequential instruction after CALL WRITE.

#### WRITE parameters:

buffer A buffer of EBCDIC text of any length.

length One word containing the number of characters in the buffer.

crlf A binary value of 1 specifies that the terminal is to be issued a carriage return and line feed (CRLF) after the message is sent. Any other value results in no CRLF being sent.

If CRLF is not equal to 1, trailing blanks in the buffer are transmitted to permit you to position the terminal cursor for the next message or operator response.

The Multiple Terminal Manager does not keep track of current terminal cursor or carriage position. No CRLF is inserted if, due to messages without CRLF or a buffer size larger than the terminal line length, the right margin is reached.

Upon completion, the contents of the buffer are unchanged.

If executed by an IBM 4978/4979/3101, control returns immediately to the caller.

No operator entry is permitted (see ACTION if operator entry is required).

## Communicating with IBM 4978/4979/3101 Displays

The Multiple Terminal Manager provides the following callable functions for specific control of the IBM 4978/4979/3101 display:

- SETPAN Retrieve a screen image
- CHGPAN Reset the unprotected character count
- SETCUR Set the cursor position
- BEEP Sound the audible alarm
- FTAB Build unprotected input field table

### CALL SETPAN

CALL SETPAN, (dsname), (return code)

This routine causes the specified screen format to be read into the Input Buffer (replacing the last operator input) and sets a switch to cause the screen format to be written to the screen during the next output cycle. Any nulls (X'00') in the screen image will be written unprotected. All other characters will be written protected. In addition to the 1920-byte screen being placed into the Input Buffer, any unprotected defaults that were specified when the screen was built, are moved, concatenated, into the Output Buffer. The cursor position for the next display after SETPAN will be set at the first unprotected character position. Before executing a CALL SETPAN, be sure to save desired information which is in the buffers, as they will be overlaid by the screen definition.

## SETPAN parameters:

dsname The data set name of the desired screen format in the SCRNS volume.

return code. A word to receive the return code. The following is a list of the possible return codes:

- -1 = Successful, new screen in buffer.
- -500 = This terminal is not an IBM 4978/4979/3101.
  No action has been taken.
- -501 = Screen data set not found.
  - 1 = Warning, data set does not contain a valid \$IMAGE screen. Input Buffer has been set to unprotected nulls (X'00') and cursor position set to (0,0).
  - 2 = Warning, too many unprotected default characters in the screen definition. The number of default characters that will be displayed has been truncated.

This return code is received if there are no default unprotected characters in the screen. The \$IMAGE utility initially assigns 1920 unprotected characters to a screen. This number is unchanged if the data (unprotected) was not modified using the edit mode of the \$IMAGE utility.

Use PF2 with \$IMAGE to enter default data.

Other = Return code from disk READ. See the <u>Language Reference</u>.

## CALL CHGPAN

CALL CHGPAN

After a CALL SETPAN, the protected characters of the screen specified have been placed in the Input Buffer. You can add data to the image by changing the Input Buffer prior to the next output cycle, and the data is displayed as protected data. If you do this, you must also CALL CHGPAN to inform the manager that it needs to recompute the location of the first unprotected character position in the current screen and the count of unprotected characters. The cursor position is set to the first unprotected character position. CHGPAN also sets the SETPAN indicator thus allowing applications to dynamically develop protected screens.

Dynamic Screen Modification and Creation: By direct manipulation of the Input and Output Buffers it is possible to modify screens built by \$IMAGE and retrieved by SETPAN. It is also possible to create screen images dynamically.

The Input Buffer contains a 24 X 80 (1920) byte image of the screen wherein unprotected fields are represented by null (zero) fields. The other bytes will be displayed as protected characters. Additional protected characters may be added to the screen image simply by inserting them in the appropriate positions in the Input Buffer. Additional unprotected fields can be added to the screen image by inserting nulls appropriately. Both protected and unprotected fields can be modified, deleted, extended, or contracted by the correct insertion of characters in the desired portions of the Input Buffer. If this is performed, it is necessary to call CHGPAN in order to indicate screen image modification.

It is also possible to modify the contents of the Output Buffer. For example, after a call to SETPAN, the Output Buffer may be modified to allow the program to modify or supply default data. Furthermore, if the Input Buffer is filled with null characters, the contents of the Output Buffer will be displayed "as is". CHGPAN must be called whenever the Input Buffer is modified.

To create a new screen, fill up the Input Buffer as desired with protected and unprotected characters, blanks, and null fields. Place default data in the Output Buffer, and call CHGPAN.

# CALL SETCUR

## CALL SETCUR, (row), (column)

CALL SETCUR specifies (overrides) the position at which the cursor is to be displayed for the next output cycle.

## SETCUR parameters:

row One-word value representing the row position, 0-23.

column One-word value representing the column position, 0-79.

The cursor position for each screen displayed on a terminal is set to first unprotected character position by default. This function permits you to override the cursor position for the output only.

### CALL BEEP

# CALL BEEP

CALL BEEP causes the audible alarm (if available) to be sounded following the next output cycle.

The IBM 4979 terminal has no audible alarm and ignores this request.

When executed for an ASCII terminal, this request causes the next output line to be followed by a bell character.

# CALL MENU

## CALL MENU

CALL MENU immediately aborts the current dialog and causes the terminal's menu screen (or request for program name message) to be displayed.

The operator can cause this at any time by pressing PF3 at an IBM 4978/4979/3101 or by typing OUT on an ASCII terminal while in a dialog.

## CALL FTAB

CALL FTAB, (table), (size), (return code)

FTAB sets up a table which describes the unprotected (input) fields placed in the Input Buffer after a SETPAN or a CHGPAN has been executed. The table is a sequence of 3-word entries which describe unprotected (input) fields. This is useful for such functions as setting the cursor.

Note: The FTAB function must be included in the application link for it to be available. See the section on "Program Preparation" on page 164 for information.

## FTAB parameters:

table

A sequence of 3-word entries which describe the unprotected fields of the screen image in the Input Buffer. Each entry contains the starting row and column positions, and the length (in bytes) of a field. Unused entries in a table will be set to zero. The format is as follows:

table (first field) row column length (second field) table+6 row column 77 length table+12 row (third field) column \*\* 77 length : : : : :

size

A word which gives the number of 3-word entries in the table.

return code A word for the return of a status code. The return codes are as follows:

-2 = FTAB code not linked with application

-1 = successful return

1 = no data fields found

2 = warning, table truncated

CALL FAN

FAN performs no operation ("no-op").

## Accessing the Terminal Environment Block

Although the terminal environment block (TEB) can be accessed directly (since its address is a user program parameter), the user program may find it more convenient with the following function to determine the attributes of the calling terminal.

### CALL CDATA

CALL CDATA, (type), (userid), (userclass), (termname), (buffersize)

This subroutine returns data concerning the terminal currently executing the program.

#### CDATA parameters:

type A word specifying the terminal type:

0 = Terminal is an IBM 4978, 4979, or 3101

2 = Terminal is an ASR 33/35 or equivalent

userid The 4-byte value set by the SIGNON program when the current terminal signed on. If the current terminal does not use SIGNON, this value is meaning-

less.

userclass The 4-byte value set by the SIGNON program when the

current terminal signed on. If the current terminal does not use SIGNON, this value is meaning-

less.

termname The 8-byte (right padded with blanks, if

necessary) name of the current terminal.

buffersize The length of the terminal's I/O buffer. For IBM 4978/4979/3101 terminals, this is the number of

unprotected characters in the last screen which

was set using SETPAN.

#### Disk File Support

All requests for disk/diskette I/O are by means of a call to the FILEIO routine. FILEIO provides the following functions:

- Automatic open of the requested data set.
- Direct access support for non-Indexed Access Method files, where records are accessed by a relative record number (RRN).
- Support for Indexed Access Method files, providing a high-level language interface to most Indexed Access Method services.
- Data integrity, via automatic close at terminal manager shutdown and automatic write back of data buffers.

If Indexed Access Method files are used, the Event Driven Executive / Indexed Access Method (5719-AM3) is required.

Automatic OPEN/CLOSE: FILEIO automatically controls opened/closed status of a data set. Thus data set names must not be coded on the PROGRAM statement of Multiple Terminal Manager programs. If the data set is not open when a request is made, the data set is opened. Since many terminals can require many data sets, both the same and different, the user can find that there was no storage available to open a requested data set. In order to avoid this situation, a limit is set for the number of open data sets. In the Multiple Terminal Manager default system, space is allocated for 14 open data sets. When this limit is reached, the least recently accessed data set is closed, and the space it required is reused. A data set is not available for automatic close if it has an update pending. The user can adjust the maximum number of open data sets by changing the file table in the Multiple Terminal Manager source module CDMCOMMN.

Indexed File Support: FILEIO provides an interface to the Event
Driven Executive Indexed Access Method.

Programs written in high-level languages can access indexed files by calling the FILEIO routine. The functions supported are listed under the heading "Indexed File Request Types" in this section. An Indexed Access Method file must be created. For information on how to create an Indexed Access Method file, see the <a href="System Guide">System Guide</a>.

Some features of the indexed file support include the follow-ing:

- Records can be retrieved sequentially or by key.
- The key can be a generic key, that is, the first n bytes of the actual key.
- Records can be added or deleted by key.
- It takes the same length of time to retrieve added records as original records.

If an application requires access to a file sequentially, and also directly by alphameric keys, indexed files are required.

Since Indexed Access Method files are owned by a supervisor task, using the ¢C command to cancel the terminal manager does not close these files. For data integrity, use the DISCONNECT, ALL command described in the section "Operator Interface" on page 158.

Additional information on indexed files and indexed file request types is discussed in the <u>System Guide</u> under "Indexed Access Method".

# CALL FILEIO

FILEIO provides the facility to access previously created files via the call interface described earlier. These files must have been previously defined and loaded.

CALL FILEIO, (fca), (buffer), (return code)

#### FILEIO parameters:

fca The file control area. The address of a table with the parameters describing the requested operations:

141

O Request Type A 4-byte EBCDIC request, for example: CL4'READ'

4 Data Set Name An 8-byte EBCDIC data set name

12 Key Relation A 2-byte EBCDIC key relation Operator operator, the characters "GT", "GE", "EQ" (indexed files only)

or
Number of A wor
Records of 25
read

A word value for the number of 256-byte records to be read or written by this call (direct files only)

14 Key Length A word specifying the length of the key to be used for retrieval. If the length specified is less than the actual key length, the first n bytes of the key are used (indexed files only).

16 Key Location The address of the key (FORTRAN, EDL, and PL/I) to be used (indexed files only).
For COBOL, the value must be 0.

or
EOD Record The system maintained logical
Number EOD record number passed back
to the application after each
direct file READ or WRITE
(direct files only).

18 Reserved

20 Relative A word value for the Record RRN. The first record is Number record number 1 (direct files only).

22 Volume Name A 6-byte EBCDIC volume name

28 Key Field The key to be used (COBOL indexed files only), if Key Length non-zero.

buffer The address of the user program I/O buffer. This is in the user program space. FILEIO and Indexed Access Method have their own buffers.

return code The address of the 2-byte field to contain the return code passed back by FILEIO. This can be a FILEIO return code, an Event Driven Executive system error code or an Indexed Access Method code.

| File Control Area (FCA): The entire FCA must be mapped for Event Driven Language, FORTRAN, PL/I, and COBOL except as noted.

0					
	REQUEST TYPE				
4					
	DATA SET NAME				
12	KEY REL OP OR NUMBER OF RECORDS				
14	KEY LENGTH				
16					
1.0	KEY LOCATION OR EOD RECORD NUMBER				
18	RESERVED				
20	RELATIVE RECORD NUMBER				
22					
	VOLUME NAME				
28					
	KEY FIELD (COBOL index files only)				
	(size defined in KEY LENGTH field)				
	///////////////////////////////////////				

Indexed File Request Types: The indexed file request types and
functions are defined as follows:

RELS Release from sequential processing mode

RELR Release a record held for update

PUTU Put operation, update mode

PUTD Put operation, delete mode

PUTN Put operation, new mode adds a record to the file

GETD Get operation, direct read

GETS Get operation, sequential read

IDEL Delete operation

ICLS Close an indexed data set

GTDU/GTRU Direct get, update mode

GTSU Sequential get, update mode

Note: GTDU and GTRU are identical in the operation they perform.

**Direct File Request Types:** The direct file request types and functions are defined as follows:

READ Read the record defined by the RRN field of the FCA into the user-provided buffer

WRIT Write the record defined by the RRN field of the FCA into the major user-provided buffer

SEOD Set the system maintained EOD pointer to the record number provided in the relative record number field of the FCA

## FILEIO Return Codes

Return	
Code	Description
-1	Successful
201	Data set not found
202	Volume not found
203	No file table entries are available; all have
	updates outstanding
204	I/O error reading volume directory
205	I/O error writing volume directory
206	Invalid function request type
	(this is returned for a valid Indexed Access
	Method function if the Indexed Access Method
	link module is not linked with the Multiple
	Terminal Manager)
207	Invalid key operator
208	SEOD record number greater than data set length

Other return codes not shown above are returned by the Indexed Access Method or by the Event Driven Executive data management support.

#### Event Driven Executive Direct File I/O Considerations

The Multiple Terminal Manager FILEIO interface to Event Driven Executive direct file support allows the user to access records by specifying relative record numbers (RRNs). Normally, a direct file may be viewed as a sequence of records starting with RRN=1 and continuing until the end of data record number, that is, RRN=EOD. The end of data record number is returned in the file control area (FCA) after each READ or WRIT (write) request. It may be set by a "set end of data" (SEOD) request.

No effort is made to ensure the data integrity of Event Driven Executive direct files involving concurrent access to the same record. That is, no record locking is performed. However, it is possible to ensure that Multiple Terminal Manager applications cannot access the same record concurrently by ensuring that application is not swapped out of the application area at an inappropriate moment. (An application is only vulnerable to swap out during an ACTION, LINK, LINKON, WRITE, or CYCLE). That is, an application can read, modify, and write a particular record and be assured that another Multiple Terminal Manager application will not alter the record at the same time.

This technique only applies to applications competing for concurrent access under a single copy of Multiple Terminal Manager. Other disciplines must be used if other applications are involved.

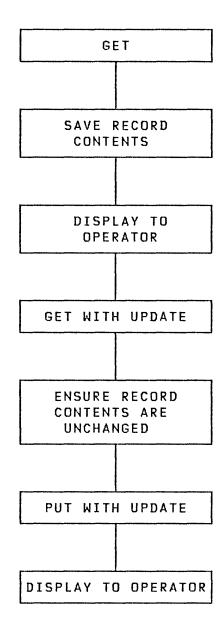
If a user desires sequential access to a direct file, it is the user's responsibility for incrementing the RRN field and ensuring it does not exceed the end of data record number. One technique involves reading the file to get the end of data record number, and then entering a loop, as in the example on the following page where a file "A,EDX002" is processed.

```
* GET EOD (RETURNED BY READ OPERATION)
        MOVE RRN,1
        CALL FILEIO, (FCA), (BUFFER), (RC)
* PROCESS FILE FROM RRN=1 TO EOD
        MOVE RRN, 0
        DO EOD, TIMES
          ADD RRN,1
          CALL FILEIO, (FCA), (BUFFER), (RC)
           . . . .
        ENDDO
        • • • •
* FILE CONTROL AREA
       EQU *
REQTYPE DATA
               CL4'READ'
DSNAME DATA
               CL8'A'
NUMREC DATA
               F'1'
               F'0'
        DATA
               F'0'
EOD
        DATA
             F'0'
        DATA
RRN
               F'0'
        DATA
VOLNAME DATA CL6'EDX002'
```

#### FILEIO Indexed Access Method Considerations

FILEIO uses the parameters provided to create a parameter list for an Indexed Access Method supervisor call. Therefore, it is important to understand Indexed Access Method operation, as explained in the section "Indexed Access Method" of the <a href="System Guide">System Guide</a>.

FILEIO executes a file cleanup routine after each user program ACTION, LINK, LINKON, WRITE, or CYCLE. If any record locks have not been released, the cleanup routine causes these records to be released in order to prevent any deadlock situations. A procedure to ensure data integrity on update is illustrated as follows:



If sequential processing has been initiated on any indexed files, the FILEIO cleanup routine also releases those files from sequential processing mode. Thus, in order to continue sequential processing from the same key, applications should save the last key before issuing an ACTION, LINK, LINKON, WRITE or CYCLE.

An indexed file may be scanned from beginning to end by use of a sequence of "get sequential" (GETS) operations. The first GETS in a sequence thereof should specify a key of all nulls (X'00') and a key relational operator of greater than (C'GT'). When executed, this initial GETS operation will receive the first record in the file (following the record, if any, for which the key is all nulls.) Subsequent GETS will retrieve the records following the first, in sequence.

After a DISCONNECT, ALL command is issued, FILEIO executes a termination routine before the Multiple Terminal Manager terminates. This termination routine closes all remaining open Indexed Access Method files. This causes any control information and records remaining in the Indexed Access Method internal storage buffers to be written to disk.

Following is a mapping of Multiple Terminal Manager/Indexed Access Method request types to the actual Indexed Access Method function.

MULTIPLE	TERMINAL	INDEXED	ACCESS
MANAGER	REQUEST	METHOD	FUNCTION
RELS		ENDSEG	)
RELR		RELEAS	SE
PUTU		PUTUP	
PUTD		PUTDE	
PUTN		PUT	
GETD		GET	
GETS		GETSEG	}
IDEL		DELETE	
ICLS		DISCON	1N
GTDU/	GTRU	GET/UF	PEQ,UPGT,UPGE
GTSU		GETSEG	)

Note: The Indexed Access Method is accessed by the Multiple Terminal Manager and, therefore, the application programs that run under the Multiple Terminal Manager will not need to include the Indexed Access Method equates and, must not be LINKed with Indexed Access Method link module.

## Programming Considerations

Multiple Terminal Manager applications are processed as initial tasks of a program which execute within the program manager's overlay area. On the first execution of a program during a transaction, the program is brought into the overlay area via a LOAD instruction. Then, when the program returns control to the Multiple Terminal Manager via a CALL ACTION, WRITE, CYCLE, MENU, LINK or LINKON, the Multiple Terminal Manager dequeues the program from Event Driven Executive via instruction. Also, if the program returned via a CALL ACTION, WRITE or CYCLE, the Multiple Terminal Manager writes the program out to the MTMSTORE data set. The overlay area is then free for use by other programs. When the Multiple Terminal Manager is ready to re-execute that program for subsequent processing of the transaction, the program manager reads the program into the overlay area and requeues that program to Event Driven Executive via an ATTACH instruction.

Thus, Multiple Terminal Manager application programs should adhere to the following conventions:

- No subtasks should be active across calls to the Multiple Terminal Manager.
- No system-wide resources should be enqueued across calls to the Multiple Terminal Manager.
- Application programs cannot use overlays.
- Application programs must be written as subroutines named MTMSUB and designed to receive four parameters at initiation.
- Application programs should utilize the Multiple Terminal Manager for all terminal and disk I/O.
- All other I/O should be complete prior to any call to the Multiple Terminal Manager.
- Application programs should terminate only via calls to the Multiple Terminal Manager and should not issue any PROGSTOP, ENDTASK, or DETACH instructions.
- Error exit routines should terminate via a CALL MENU.
- Changes affecting the SCRNS or PRGRMS volumes during the Multiple Terminal Manager session will not be effective until the Multiple Terminal Manager is terminated and reloaded.

## Event Driven Language Programming Considerations

An Event Driven Language application, which must be written as a subroutine, must be defined to accept four parameters. In addition, the Multiple Terminal Manager functions must be identified via the EXTRN statement. The subroutine name MTMSUB must also appear on the ENTRY statement. For example:

ENTRY MTMSUB
EXTRN ACTION, BEEP, CYCLE, SETCUR, CHGPAN, CDATA, MENU
EXTRN SETPAN, FILEIO, LINK, LINKON, WRITE, FTAB, FAN
SUBROUT MTMSUB, INPUT, OUTPUT, TEB, IIB

The interface used by the Multiple Terminal Manager stub CDMEMAIN for calling the Event Driven Language subroutine is via the CALL instruction.

For example, the statement to call SETPAN is:

CALL SETPAN, (MENUNAME), (RC)

This statement would result in the addresses of MENUNAME and RC being passed to the Multiple Terminal Manager.

The syntax for calling Multiple Terminal Manager functions in the Event Driven Language is:

CALL ACTION

CALL ACTION, (BUFFER), (LENGTH), (CRLF)

CALL LINK, (PROGRAM)

CALL LINKON, (PROGRAM)

CALL CYCLE

CALL WRITE, (BUFFER), (LENGTH), (CRLF)

CALL SETPAN, (DSNAME), (RC)

CALL CHGPAN

CALL SETCUR, (ROW), (COLUMN)

CALL BEEP

CALL MENU

CALL CDATA, (TERMTYPE), (USERID), (USERCLASS), (TERMNAME), (BUFSIZ)

CALL FILEIO, (FCA), (BUFFER), (RC)

CALL FAN

## FORTRAN Programming Considerations

A FORTRAN application, which must be written as a subroutine, must be defined to accept four parameters, for example:

### SUBROUTINE MTMSUB(INPUT,OUTPUT,TEB,IIB)

The interface used by the Multiple Terminal Manager stub CDMFMAIN for calling the FORTRAN subroutine is via the Event Driven Language CALLFORT instruction. For interfacing to the Multiple Terminal Manager, FORTRAN applications utilize the FORTRAN CALL statement for calling Event Driven Executive subroutines.

For example, the statement to call SETPAN is:

```
CALL EDX(SETPAN, 2, IADDR(MENUNAME), IADDR(RC))
```

This statement would result in the addresses of MENUNAME and RC being passed to the Multiple Terminal Manager.

All Multiple Terminal Manager functions which the application calls must be declared as EXTERNAL, for example:

### EXTERNAL SETPAN, ACTION, MENU, FILEIO

The syntax for calling Multiple Terminal Manager functions in FORTRAN is:

```
CALL EDX(ACTION, 0)
CALL EDX(ACTION, 3, IADDR(BUFFER), IADDR(LENGTH), IADDR(CRLF))
CALL EDX(LINK,1, IADDR(PROGRAM-NAME))
CALL EDX(LINKON, 1, IADDR(PROGRAM-NAME))
CALL EDX(CYCLE,0)
CALL EDX(WRITE, 3, IADDR(BUFFER), IADDR((LENGTH), IADDR(CRLF))
CALL EDX(SETPAN, 2, IADDR(DSNAME), IADDR(RET-CODE))
CALL EDX(CHGPAN,0)
CALL EDX(SETCUR, 2, IADDR(ROW), IADDR(COLUMN))
CALL EDX(BEEP,0)
CALL EDX(MENU,0)
CALL EDX(CDATA,5,IADDR(TERM-TYPE),IADDR(USERID),
           IADDR(USER-CLASS), IADDR(TERM-NAME), IADDR(BUF-SIZE))
CALL EDX(FILEIO, 3, IADDR(FILE-CONTROL-AREA), IADDR(BUFFER),
           IADDR(RET-CODE))
CALL EDX(FTAB, 3, IADDR(TABLE), IADDR(SIZE), IADDR(RC))
CALL EDX(FAN, 0)
```

## COBOL Programming Considerations

The PROGRAM-ID for all Multiple Terminal Manager COBOL applications must be "MTMSUB". In addition, all parameters passed to the Multiple Terminal Manager must be level 01 or 77. The four parameters passed to the application, Input Buffer, Output Buffer, TEB, and IIB must be defined in the program's LINKAGE SECTION. Refer to "COBOL Sample Prog1" on page 193 for an example. The PROCEDURE DIVISION must contain the USING clause followed by the names given to the Input Buffer, Output Buffer, TEB, and IIB, in that order.

For CALL FILEIO, if key location equals 0 and key length not equal to 0, the file manager assumes that the key is immediately following the FCA. This is primarily to facilitate COBOL programs, which cannot code addresses.

The following example shows an FCA for indexed files which would read a record associated with a 4-character key "XXXX".

- 01 FILE-CONTROL-AREA.
  - REQUEST-TYPE PIC X(4) VALUE "GETD". 05
  - DATA-SET-NAME PIC X(8). 05
  - PIC XX VALUE "EQ". 05 KEY-REL-OP
  - PIC S999 COMP VALUE 4. 05 KEY-LENGTH
  - PIC S999 COMP VALUE O. KEY-LOCATION 05
  - 05 FILLER PIC X(4).
  - PIC X(6). 05 VOLUME-NAME
  - PIC X(4) VALUE "XXXX". 0.5 KEY

For interfacing to the Multiple Terminal Manager, COBOL applications utilize the COBOL CALL statement for calling subroutines.

For example the statement to call SETPAN is:

CALL "SETPAN" USING SCREEN, RC.

This would result in the addresses of SCREEN and RC being passed to Multiple Terminal Manager.

The WORKING-STORAGE SECTION would have the following:

- SCREEN PICTURE X(8) VALUE "SCRNNAME".
- RC PICTURE 99 COMP. 77

The syntax for calling Multiple Terminal Manager functions in COBOL is:

CALL "ACTION".

CALL "ACTION" USING BUFFER, LENGTH, CRLF.

CALL "LINK" USING PROGRAM-NAME.

CALL "LINKON" USING PROGRAM-NAME.

CALL "CYCLE".

CALL "WRITE" USING BUFFER, LENGTH, CRLF.

CALL "SETPAN" USING DATA-SET-NAME, RETURN-CODE.

CALL "CHGPAN".

CALL "SETCUR" USING ROW, COLUMN.

CALL "BEEP".

CALL "MENU".

CALL "CDATA" USING TERMINAL-TYPE, USER-ID, USER-CLASS,

CALL "FILEIO" USING FILE-CONTROL-AREA, BUFFER, RETURN-CODE.

TERMINAL-NAME, BUFFER-SIZE.

CALL "FTAB" USING TABLE, SIZE, RETURN-CODE.

154 SC34-0316

CALL "FAN".

A PL/I application must be named MTMSUB, and defined to accept four parameters:

```
MTMSUB: PROCEDURE (INPUT_BUFFER, OUTPUT_BUFFER, TEB, PF_KEY);
```

INPUT\_BUFFER, OUTPUT\_BUFFER, and TEB should usually be declared as structures. PF\_KEY should be declared BINARY FIXED (15).

All Multiple Terminal Manager functions which the application calls must be declared as ENTRY, for example:

```
DECLARE
(SETPAN, ACTION, MENU, SETCUR, BEEP, FILEIO)
ENTRY;
```

The syntax for calling Multiple Terminal Manager functions in PL/I is:

```
CALL ACTION;
CALL ACTION(BUFFER, LENGTH, CRLF);
CALL LINK(PROGRAM_NAME);
CALL LINKON(PROGRAM_NAME);
CALL CYCLE;
CALL WRITE(BUFFER, LENGTH, CRLF);
CALL SETPAN(DATA_SET_NAME, RETURN_CODE);
CALL CHGPAN;
CALL SETCUR(ROW, COLUMN);
CALL BEEP;
CALL MENU;
CALL CDATA(TERMINAL_TYPE, USER_ID, USER_CLASS,
           TERMINAL_NAME, BUFFER_SIZE);
CALL FILEIO(FILE_CONTROL_AREA, BUFFER, RETURN_CODE);
CALL FTAB(TABLE, SIZE, RETURN_CODE);
CALL FAN;
```

For WRITE, the buffer variable must be a character string. For FTAB, the table variable must be an array. All variables should be declared as STATIC whenever possible.

#### SIGNON

A sample SIGNON program is distributed with the Multiple Terminal Manager. If the terminal requires sign-on, the IBM supplied SIGNON program displays the SIGNON screen and does a CALL ACTION to obtain the user ID and password.

The user must enter the sign-on ID (8 bytes alphanumeric) and a password (4 bytes alphanumeric). This data will be passed to the SIGNON program in the Input Buffer as it would be to any other program. The sign-on ID and password are validated against the SIGNON file. If valid, the sign-on is complete and the primary menu is displayed. If invalid, a bad return code is set (=1) and the SIGNON program is reloaded by Multiple Terminal Manager. The two sign-on ID records in the distributed SIGNON file are:

SIGNON ID	PASSWORD
11111111	1111
2222222	2222

You can add additional records with the Event Driven Executive text editor.

In addition to the four parameters passed to all applications, the SIGNON routine receives a fifth parameter which is the address of the sign-on control area. The contents of the sign-on control area are as follows:

- RC 2-byte return code indicating to the system the action to be taken.
- USERID Four bytes handled exactly like USERCLASS.
- USERCLASS Four bytes set by user sign-on program which will be saved and passed as a parameter to the sign-off program when the current user signs off. These four bytes are contained in the TEB and are also available to any standard program to validate the user if desired.
  - O = valid sign-on, display the terminal's menu screen.
  - 1 = invalid sign-on, redisplay the sign-on screen.

USERCLASS and USERID are not used by the Multiple Terminal Manager. They are saved in the TEB and reported via CALL CDATA to requesting programs from this terminal while the current sign-on is active.

### SIGNOFF

A sign-off program is not provided with the default system; however, provisions are made within the Multiple Terminal Manager to invoke a sign-off program. If you write a sign-off program, it will be passed the same parameters as the sign-on program.

If these programs exist, they must meet the following considerations:

- SIGNON and SIGNOFF are optional. Either SIGNON alone or SIGNON and SIGNOFF can be in the system. If they are in the system, the names must be SIGNON and SIGNOFF. If they are not in the system, the names SIGNON and SIGNOFF must not be used for other user-written programs.
- The SIGNOFF program is invoked when the PF3 key is entered from the menu screen.
- SIGNON/SIGNOFF cannot be executed from the menu screen by entering the program name.
- Individual terminals can be generated to require or not require sign-on. If the user does not include a SIGNON program, any terminals marked requiring sign-on are unusable since there is no way to validate sign-on attempts.
- SIGNON/SIGNOFF can use CALL CDATA to obtain the terminal name and other terminal information.
- When complete, SIGNON/SIGNOFF should perform a CALL MENU to return to the Multiple Terminal Manager. Note that a return code should be set in the RC field by the SIGNON program before issuing the CALL MENU. The RC field is ignored by the Multiple Terminal Manager for the SIGNOFF program.
- The use of USERCLASS and USERID is optional.
- LINK and LINKON can not be used.
- PF3 entered by the operator during SIGNON, will cause the current SIGNON session to be terminated and a new SIGNON session to be started.

# Multiple Terminal Manager Initiation and Termination

The Multiple Terminal Manager can be initiated from any terminal defined to the Event Driven Executive system by entering the \$L \$MTM,PRGRMS command. This command starts the Multiple Terminal Manager program manager. The program manager then initiates a terminal server for each terminal specified in the TERMINAL file. Upon completion of initiation, the IPL screen, IPLSCRN, is displayed at each of the Multiple Terminal Manager terminals. IPLSCRN specifies that the operator press the ENTER key in order to display either the sign-on or menu screen.

The Multiple Terminal Manager is terminated by disconnecting all terminals using the DISCONNECT command. The \$C command should not be used to terminate Multiple Terminal Manager tasks.

### Signing On

If sign-on is specified for the terminal, then the sign-on screen, SIGNON, is displayed following the IPL screen. The sign-on screen requires that the operator enter a sign-on and password. After sign-on processing is completed, the menu screen is displayed.

## Program Initiation and Termination

After Multiple Terminal Manager initiation and sign-on processing are completed, the menu screen is displayed. The menu screen is the screen from which the operator can initiate transactions. A transaction is initiated by the operator entering either a program name or pressing a PF key when the menu screen is displayed. A PF key initiates program PFnn, where nn reflects the number of the PF key pressed. If data is entered, the Multiple Terminal Manager considers the first eight bytes to be a program name.

After a transaction is initiated, the operator can terminate it by pressing the PF3 key. Upon termination of the transaction, the menu screen is redisplayed. A subsequent pressing of the PF3 key from the menu screen causes the sign-on screen to be redisplayed if sign-on is specified for that terminal. Otherwise, PF3 will be a "no-op" and the menu screen remains displayed.

#### Utilities

Disconnect: Terminals can be disconnected from the Multiple Terminal Manager or the Multiple Terminal Manager can be terminated via the DISCONNECT facility. DISCONNECT is invoked from the menu screen by keying in either DISCONNECT, DISCONNECT \*, DISCONNECT, termname, or DISCONNECT, ALL. If DISCONNECT or DISCONNECT \* is entered, the terminal upon which that request was entered is disconnected. If a referenced terminal is in a transaction, that transaction is allowed to complete. When the terminal returns to MENU state, it is automatically signed off and immediately displays the YOU ARE DISCONNECTED message.

If DISCONNECT, ALL is specified, all terminals are disconnected. When the last terminal is truly disconnected, whether via DISCONNECT, ALL or separate DISCONNECTs, the manager task is stopped. This is the only method that should be used to terminate the Multiple Terminal Manager.

Note that to enter this command from a screen, the terminal's menu screen must contain at least 19 unprotected characters.

While a terminal continues in a transaction with disconnect pending, the audible alarm is sounded after every interaction to tell the operator that a disconnect is pending.

**Reconnect:** If the referenced terminal is disconnected, it is reconnected using RECONNECT, ALL or RECONNECT, termname in a signed-off status (if applicable). If the terminal is not disconnected, the command is ignored. The reconnect should be issued from a terminal other than the disconnected terminal. The program name of this command is RECONNEC.

**Programs Report:** This report displays data about each available program. It is intended mainly for debugging during development of the manager but is included as a working example for possible use.

The name of this program is PGMRPT.

The Programs Report will have the following headings:

PGM NAME LENGTH (in records)

**Terminal Activity Report:** This program displays the names and status of all terminals on the system. If more than 19 terminals are attached, the operator must press ENTER to page to successive groups of 19 lines.

The name of this program is REPORT.

The Terminal Activity Report has the following headings:

TERMINAL TERMINAL USER USER PROGRAM OPERATOR TERMINAL NAME TYPE ID CLASS INPUTS OUTPUTS

Screens Report: This program displays the names of the screens defined in the SCRNS volume. The operator can key in the screen name to be displayed.

The name of this program is SCRNSRPT.

**Screen Print:** Displayed screens on a 4978 or 4979 terminal can be printed on the system printer by pressing the PF6 key or the key specified on the HDCOPY parameter of the TERMINAL statement during system generation.

### DISTRIBUTION, INSTALLATION AND PROGRAM PREPARATION

The Multiple Terminal Manager is distributed as a program product and each distribution consists of the following items:

- Prebuilt Multiple Terminal Manager This is a prebuilt Multiple Terminal Manager consisting of a program manager, file manager, terminal servers and utility programs. The Indexed Access Method interface is not included.
- Multiple Terminal Manager source for module CDMCOMMN -This is the Multiple Terminal Manager source code for the user who wants to tailor the Multiple Terminal Manager environment.
- Screen formats This is a set of screens to support the default Multiple Terminal Manager and sample programs.
- TERMINAL File This is a set of miscellaneous terminal statements to support the default system.
- CDMEMAIN, CDMFMAIN, CDMCMAIN, and CDMPMAIN These are the Multiple Terminal Manager application stubs in object format that must be included with either Event Driven Language, FORTRAN, COBOL, or PL/I programs at link time.

#### Installation

The user must have created the following volumes on the system disk at system generation time.

PRGRMS This volume is for the Multiple Terminal Manager programs, user application programs, terminal specifications file and SIGNONFL file.

SCRNS This volume is for the screen formats used by Multiple Terminal Manager and user applications.

MTMSTR This volume is for the MTMSTORE data set used by the Multiple Terminal Manager.

After the volumes have been created, the user can then copy the prebuilt Multiple Terminal Manager, screen formats and terminal file from the source diskettes to disk. This installs the default Multiple Terminal Manager and establishes the following data sets.

Data sets within the PRGRMS volume:

\$MTM The Multiple Terminal Manager program manager

CDMSVR89 The Multiple Terminal Manager full screen, 4978 and 4979, terminal server

CDMSVR33 The Multiple Terminal Manager TTY terminal server

CDMSVR01 3101 Model 2 terminal server

CDMINIT The Multiple Terminal Manager initialization routine

TERMINAL The Multiple Terminal Manager terminal specification file

In addition, the PRGRMS volume contains miscellaneous data sets needed for the utility programs.

Data sets within the SCRNS volume:

IPLSCRN The initial Multiple Terminal Manager displayed screen

SIGNONSC The sign-on screen

MENUSCRN The default menu screen

SCRNSREP The SCRNSRPT selection menu

The Multiple Terminal Manager can be tailored by reassembling, rebuilding and replacing the changed Multiple Terminal Manager components.

The terminal specifications file (TERMINAL) can be modified to match your system environment by using the \$FSEDIT Event Driven Executive utility. Screen formats can be added to the SCRNS volume via the \$IMAGE Event Driven Executive utility.

Before executing the Multiple Terminal Manager, the user has to create the MTMSTORE dataset.

## Event Driven Language Program Preparation

The Multiple Terminal Manager contains a main routine, CDMEMAIN, for supporting Event Driven Language applications. CDMEMAIN is the Multiple Terminal Manager stub for Event Driven Language applications, and is object code which enables the Multiple Terminal Manager to invoke and pass parameters to the application program.

It is necessary to link CDMEMAIN with the application object module so that the application can communicate with the Multiple Terminal Manager. For linking Event Driven Language applications, this requires that the following be used as the link control data set during the \$LINK program preparation step:

> OUTPUT output data set, volume INCLUDE CDMEMAIN, volume INCLUDE object data set, volume END

For example, the link control statements for an Event Driven Language application called "QUERY" might be:

> OUTPUT QUERY, EDX002 INCLUDE CDMEMAIN, EDX003 INCLUDE QUERY, EDX003 END

The subsequent \$UPDATE step would then specify the object input to be "QUERY,EDX002" and the program output to "QUERY,PRGRMS", where "PRGRMS" is the Multiple Terminal Manager program volume.

Note: If the FTAB function is used by the application, the FTAB object code must be linked with the application object code. This requires that the object module CDMFTAB be included in the linking process. The following link control statement must be included in the link control data set:

INCLUDE CDMFTAB, volume For example: OUTPUT QUERY, EDX002 INCLUDE CDMEMAIN, EDX003 QUERY, EDX003

CDMFTAB, EDX003

INCLUDE END

INCLUDE

## FORTRAN Program Preparation

The Multiple Terminal Manager contains a main routine, CDMFMAIN, for supporting FORTRAN applications. CDMFMAIN is the Multiple Terminal Manager stub for FORTRAN applications, and is object code which enables the Multiple Terminal Manager to invoke and pass parameters to the application program.

It is necessary to link CDMFMAIN with the application object module so that the application can communicate with the Multiple Terminal Manager. For linking FORTRAN applications, this requires that the following be used as the link control data set during the \$LINK program preparation step:

> OUTPUT output data set, volume AUTO=FORTAUTO, ASMLIB INCLUDE CDMFMAIN, volume INCLUDE object data set, volume

For example, the link control statements for a FORTRAN application called "QUERY" might be:

OUTPUT QUERY, EDX002 AUTO=FORTAUTO, ASMLIB CDMFMAIN, EDX003 INCLUDE QUERY, EDX003 INCLUDE END

The subsequent \$UPDATE step would then specify the object input to be "QUERY,EDX002", and the program output to be "QUERY,PRGRMS", where "PRGRMS" is the Multiple Terminal Manager program volume.

Note: If the FTAB function is used by the application, the FTAB object code must be linked with the application object code. This requires that the object module CDMFTAB be included in the linking process. The following link control statement must be included in the link control data set:

For example:

INCLUDE

OUTPUT QUERY, EDX002 AUTO=FORTAUTO, ASMLIB INCLUDE CDMFMAIN, EDX003 INCLUDE QUERY, EDX003 INCLUDE CDMFTAB, EDX003 END

CDMFTAB, volume

# COBOL Program Preparation

The Multiple Terminal Manager contains a main routine, CDMCMAIN, for supporting COBOL applications. CDMCMAIN is the Multiple Terminal Manager stub for COBOL applications, and is object code which enables the Multiple Terminal Manager to invoke and pass parameters to the application program.

It is necessary to link CDMCMAIN with the application object module so that the application can communicate with the Multiple Terminal manager. For linking COBOL applications, this requires that the following be used as the link control data set during the \$LINK program preparation step:

OUTPUT output data set, volume AUTO=COKAUTO, ASMLIB INCLUDE CDMCMAIN, volume INCLUDE MTMSUB#1, volume INCLUDE MTMSUB#B, volume END

In the previous example, MTMSUB#1 is the name of the data set containing the COBOL compiled output. MTMSUB#B is the name of the data set containing the COBOL I/O buffers (if required).

For example, the link control statements for a COBOL application called "QUERY" might be:

OUTPUT QUERY, EDX002 AUTO=COKAUTO, ASMLIB
INCLUDE CDMCMAIN, EDX003
INCLUDE MTMSUB#1, EDX003
END

The subsequent \$UPDATE step would then specify the object input to be "QUERY,EDX002", and the program output to be "QUERY,PRGRMS", where "PRGRMS" is the Multiple Terminal Manager program volume.

Note: If the FTAB function is used by the application, the FTAB object code must be linked with the application object code. This requires that the object module CDMFTAB be included in the linking process. The following link control statement must be included in the link control data set:

INCLUDE CDMFTAB, volume

#### For example:

OUTPUT QUERY, EDX002 AUTO=COKAUTO, ASMLIB
INCLUDE CDMCMAIN, EDX003
INCLUDE MTMSUB#1, EDX003
INCLUDE CDMFTAB, EDX003
END

# PL/I Program Preparation

The Multiple Terminal Manager contains a main CDMPMAIN, for supporting PL/I applications. CDMPMAIN is the Multiple Terminal Manager stub for PL/I applications, and is object code which enables the Multiple Terminal Manager to invoke and pass parameters to the application program.

It is necessary to link CDMPMAIN with the application object module so that the application can communicate with the Multiple Terminal Manager. For linking PL/I applications, this requires that the following be used as the link control data set during the \$LINK program preparation step:

> OUTPUT output data set, volume AUTO=PLIAUTO, ASMLIB INCLUDE CDMPMAIN, volume INCLUDE object data set, volume

For example, the link control statements for a PL/I application called "QUERY" might be:

> OUTPUT QUERY, EDX002 AUTO=PLIAUTO, ASMLIB INCLUDE CDMPMAIN, EDX003 QUERY, EDX003 INCLUDE END

The subsequent \$UPDATE step would then specify the object input to be "QUERY,EDX002", and the program output to "QUERY,PRGRMS", where "PRGRMS" is the Multiple Terminal Manager program volume.

Note: If the FTAB function is used by the application, the FTAB object code must be linked with the application object code. This requires that the object module CDMFTAB be included in the linking process. The following link control statement must be included in the link control data set:

INCLUDE CDMFTAB, volume

For example:

OUTPUT QUERY, EDX002 AUTO=PLIAUTO, ASMLIB INCLUDE CDMPMAIN, EDX003 QUERY, EDX003 INCLUDE INCLUDE CDMFTAB, EDX003 END

### STORAGE REQUIREMENTS

Listed below are the storage requirements for the Multiple Terminal Manager. These requirements are in addition to the storage required for the Multiple Terminal Manager application programs, the Event Driven Executive supervisor, the supervisor's required device support programs and control blocks.

Program manager - 12K (K = 1024 bytes)

Terminal server - 1K per terminal for TTY (ASCII)

.75K per 4978/4979 display

1.75K per 3101 Model 2 display

The storage required for Multiple Terminal Manager application programs is the larger of 6K or the size of the largest application which includes the application stub. This is the size obtained after linking the application via \$LINK.

During system configuration, the above information is used to calculate the partition size to code on the SYSTEM statement, PARTS= operand. For more information on the SYSTEM statement see the <a href="System Guide">System Guide</a>.

#### SYSTEM GENERATION CONSIDERATIONS

#### Volume Requirements

Three volumes must be provided when planning your Event Driven Executive system for the Multiple Terminal Manager. These volumes are:

- PRGRMS Multiple Terminal Manager programs volume
- MTMSTR Multiple Terminal Manager work volume
- SCRNS Multiple Terminal Manager screens volume

In Multiple Terminal Manager only systems, the most likely access frequency distribution of these three volumes would be:

- (1) MTMSTR
- (2) SCRNS
- (3) PRGRMS

Therefore, it is recommended that these volumes be allocated so that the MTMSTR and SCRNS volumes are adjacent to each other with PRGRMS on one side or the other.

If fixed-head disks are to be used, it may be beneficial to allocate the MTMSTR volume under the fixed head. (In this case the location of SCRNS relative to MTMSTR is irrelevant.) This is accomplished by specifying FHVOL=MTMSTR on the appropriate system configuration DISK statement. It will not be possible to place MTMSTR under the fixed head if the total volume size exceeds 480 records for a 4962, or 512 records for a 4963.

To calculate the size requirements for each of the three volumes, first calculate the data set requirements (see the section "Data Set Requirements" on page 171). Add to this the directory size in number of records, each volume requires. The directory sizes may be calculated as follows:

Volume MTMSTR				
SCRNS	(number of screens + 2) / 8 rounded to the next highest record			
PRGRMS	(number of user programs + 9) / 8 rounded to the next highest record			

See the <u>System Guide</u> for a sample Multiple Terminal Manager system configuration.

#### Data Set Requirements

### MTMSTORE

MTMSTORE is the Multiple Terminal Manager work file, and as such, it contains:

- The Multiple Terminal Manager program table.
- The Multiple Terminal Manager screen table.
- A program and buffer save area for each terminal defined in the TERMINAL file.

The size of the MTMSTORE file can be calculated as follows:

- Allow 10 bytes per screen in the SCRNS volume; round up to the nearest 256-byte record.
- Allow 14 bytes per program in the PRGRMS volume; round up to the nearest 256-byte record.
- Allow per terminal:

enough records to hold a copy of the largest program in the PRGRMS volume plus 4 records; round up to the nearest track; that is, nearest 64 records for a 4963 disk or nearest 60 records for a 4962 disk.

This data set is in the volume MTMSTR and is normally the only data set in that volume.

### TERMINAL

This file is built with the \$FSEDIT Event Driven Executive utility. It contains one record/terminal containing the specifications of a terminal.

The record prototype is:

Dvtp, Termname, Menuscrn, Y/N

The following is a description of the record:

Dvtp The type of terminal. Specify one of the following per terminal:

4979	(IBM	4979 full screen)	
4978	(IBM	4978 full screen)	
3335	(ASR	33/35 line at a time)	
3101	(IBM	3101 Model 2 in block	mode)

Termname The 1 to 8 character name of the terminal. This name must be identical with the device name specified on the TERMINAL statement at system generation. This name should not be the name of the Event Driven Executive \$SYSLOG device.

Menuscrn The name of the data set in the SCRNS volume which contains the screen to be displayed after an operator exits a transaction or signs on. For ASCII terminals, this field is ignored.

Y/N Specifies whether the terminal uses SIGNON/SIGNOFF.

Y = This terminal is required to use the SIGNON and SIGNOFF programs. If a user program named SIGNON does not appear in the program library, this terminal is not usable.

N = This terminal is always signed on.

Comment records are acceptable in this file as well as comments following specification records. Comment records must have an \* in position 1.

An example of this file would be:

3101, DIS31010, MENUSCRN, N 4979, DISPLAY1, MENUSCRN, N 4978, DIS49780, MENUSCRN, Y 3335, ACCA1, MENUSCRN, Y /\*

End of specifications must be indicated with a record containing /\* beginning in column 1.

Before the Multiple Terminal Manager processes each record during startup, the record is listed on the \$SYSPRTR device. When startup is complete, all terminals will have the Multiple Terminal Manager IPL screen displayed. The TERMINAL file is in the volume PRGRMS.

### <u> Screen Format Volume - SCRNS</u>

This volume contains screen data sets for full screen images built via the \$IMAGE Event Driven Executive utility. These screens must have been built with a 24 x 80 dimension size. The unprotected fields must be initialized with blanks or default data. If a screen is modified or added to the SCRNS volume, the Multiple Terminal Manager should be terminated and restarted so that the Multiple Terminal Manager can initialize linkage to the screens.

The IPLSCRN data set is displayed on each full screen terminal after the Multiple Terminal Manager is started. It requests that the operator press the ENTER key to connect the terminal to the Multiple Terminal Manager. It should not be displayed again.

Screen definition procedure (under \$IMAGE) should always be concluded by entering unprotected field initialization mode using PF2, even when a fully protected screen is being defined.

### User Application Program Volume - PRGRMS

All programs loaded by the Multiple Terminal Manager are loaded using the names of the data sets in this volume. The TERMINAL and SIGNONFL files are also in this volume.

Application programs are stored in this volume as the output of the \$UPDATE Event Driven Executive utility. The names of the programs are the names used by the operator from the MENU mode to invoke programs and can also be used as the program parameter on a CALL LINK or CALL LINKON that passes control from one program to another. (If an existing program is modified or a new program added, the Multiple Terminal Manager should be terminated and restarted so that Multiple Terminal Manager can establish linkage to these changes or additions.)

When the Multiple Terminal Manager is initiated, a program table is built containing the name of each program data set in the PRGRMS volume.

Each program is checked at initialization time to see if the program is too big for the program area in the Multiple Terminal Manager. If the program is too big for the program area in the Multiple Terminal Manager, split the program into separate programs using LINK or increase the size of the program area.

## SIGNONFL

This file contains sign-on records for use by the SIGNON program. The format of the file is:

Field Name	<u>Positions</u>	<u>Contents</u>
SIGNON ID	1-08	Sign-on ID number
PASSWORD	9-12	Password
USERID	13-16	User ID
USER CLASS	17-20	User Class
NAME	21-32	User Name

This file is built by using the \$FSEDIT Event Driven Executive utility. This file is in the volume PRGRMS. A /\* in columns 1 and 2 denote the end of the file.

# Multiple Terminal Manager Data Set Requirements for Execution

	ITEM	VOL ID	DATA SET NAME	APPROXIMATE SIZE
	SWAP DATA SET	MTMSTR	MTMSTORE	See MTMSTORE in the Multiple Terminal Manager Data Set Requirements section
	PROGRAM MGR	PRGRMS	\$MTM	55 records
	4978/4979 TERM SERVER	PRGRMS	CDMSVR89	4 records
ļ	3101 TERM SERVER	PRGRMS	CDMSVR01	8 records
	TTY TERM SERVER	PRGRMS	CDMSVR33	5 records
	MULTIPLE TERMINAL MANAGER INITIALIZATION	PRGRMS	CDMINIT	29 records
	TERMINAL SPECIFICATIONS FILE	PRGRMS	TERMINAL	1 record per 2 entries
	USER APPLICATION PROGRAMS	PRGRMS	? ?	?
	SCREEN FORMATS	SCRNS	USER SPECIFIED SCREENS	) 4 records per screen
	SIGNON FILE	PRGRMS	SIGNONFL	1 record per 2 entries

## Multiple Terminal Manager Requirements for Program Preparation

MULTIPLE TERMINAL MANAGER STUBS:

CDMFTAB

Approximate size
6 records each

CDMEMAIN CDMFMAIN CDMCMAIN

CDMPMAIN 2 records

plus Event Driven Executive program preparation data set requirements

# Requirements for Rebuilding the Multiple Terminal Manager

MULTIPLE TERMINAL MANAGER OBJECT

Approximate size 100 records

Multiple Terminal Manager source module: CDMCOMMN plus Event Driven Executive program preparation data set requirements

98 records

### MULTIPLE TERMINAL MANAGER DEFAULTS AND HOW TO CHANGE

The Multiple Terminal Manager default system contains the following limitations.

Maximum number of screens - 307

This number can be increased by increasing the Input or Output Buffer size 10 bytes per additional screen. The Input Buffer (COMINPUT) and the Output Buffer (COMOUTPT) are in the module CDMCOMMN.

Maximum number of concurrently open data sets - 14

This number can be changed by altering the file table size. The file table is in the module CDMCOMMN.

Maximum number of terminals - 10

This number can be increased by increasing the terminal table size 12 bytes per terminal. The terminal table (COMTERM) is in the module CDMCOMMN.

Maximum program size - 16K bytes

This size can be changed by reallocating the CDMDUMMY module to the desired size or by patching the name of your largest application program into the PGM1 name position of the program manager's program header. The offset in \$MTM of the name CDMDUMMY is X'D8'

 Maximum packed screen format size as built by the Event Driven Executive screen formatter, \$IMAGE - 1024 bytes

This size can be increased by increasing the screen buffer size. The screen buffer (COMPMGR) is in the module CDMCOMMN.

Maximum number of programs - 73

This number can be increased by increasing the screen buffer size 14 bytes per program. The screen buffer (COMPMGR) is in the module CDMCOMMN.

Whenever the source module CDMCOMMN is changed, it must be reassembled and the program manager must be rebuilt with the new CDMCOMMN object module.

<u>Note</u>: Changes to the screen buffer or Input Buffer must be in increments of 256 to facilitate Event Driven Executive disk READs.

### MULTIPLE TERMINAL MANAGER MESSAGES

NO TERMINALS ARE AVAILABLE: No valid terminal specification records found in the TERMINAL file, or, no terminal servers can be loaded, or, all terminals are busy. Other messages generated indicate the problem area. The manager program is terminated.

MTMSTORE DATA SET LIMITS EXCEEDED: The specified MTMSTORE file is too small. Delete and recreate it larger. The manager has been terminated.

This can occur after adding a new program with a storage requirement greater than any previous program's requirement or after adding a new terminal or screen.

**PROGRAM AREA TOO SMALL TO HOLD PGM BBBBBBBB:** The manager's program area is too small to hold the named program. The program is unusable.

Increase the program area size by reallocating CDMDUMMY or split the program into smaller LINKed programs.

BBBBBBB PROGRAM TYPE INVALID: The named program in the PRGRMS volume is not a program type data set. The named program is unusable.

SIGNON PROGRAM NOT AVAILABLE FOR TERMINAL BBBBBBBB: The specified terminal is required to sign on and off but no program named SIGNON was found in the PRGRMS volume. The terminal is not connected to the Multiple Terminal Manager.

MULTIPLE TERMINAL MANAGER TERMINAL FILE RECORDS: The TERMINAL file records processed by the Multiple Terminal Manager are listed after this message. Any messages pertaining to a specific TERMINAL file record will be displayed immediately after the file record.

**DEVICE TYPE INVALID:** The device type specified for the TERMINAL file record listed immediately before this message is invalid. The terminal is not connected. Correct the TERMINAL record. Stop and restart the manager.

INVALID SIGNON CHARACTER: The SIGNON specification for the TERMINAL file record listed immediately before this message is not "Y" or "N". The terminal is not connected. Correct the TER-MINAL record. Stop and restart the manager.

MENUNAME INVALID: The primary menu name specified for the TER-MINAL file record listed immediately before this message is invalid. The terminal is not connected. Correct the TERMINAL record. Stop and restart the manager.

TERMINAL BBBBBBBB NOT DEFINED IN EVENT DRIVEN EXECUTIVE SYSTEM: The specified terminal was not included in the definition of terminals when the Event Driven Executive system was generated. The terminal is not connected. Include a terminal definition for the specified terminal when the Event Driven Executive system is generated.

TERMINAL NAME INVALID: The terminal name specified for the TER-MINAL file record listed immediately before this message is invalid. The terminal is not connected. Correct the TERMINAL record. Stop and restart the manager.

CONNECTED TO MULTIPLE TERMINAL MANAGER: This message is written to a non-full screen type terminal when it is connected to the Multiple Terminal Manager.

LOAD FOR SERVER BBBBBBBB FAILED, RC=CCCCC: A load failure occurred during initialization for the specified server. Refer to Event Driven Executive messages and codes to determine the cause of failure. Ensure that the specified server program is in the PRGRMS volume.

PRIMARY MENU BBBBBBBB FAILED FOR TERMINAL BBBBBBBBB: A SETPAN function for the primary menu indicated has failed. Ensure that a valid menu name is specified in the TERMINAL file for the specified terminal.

DISK ERROR DURING INITIALIZATION, RC=CCCCC: A disk error occurred while reading the SCRNS volume directory, the PRGRMS volume directory, or the TERMINAL data set. Or, an error occurred while writing to the MTMSTORE data set. Determine the cause using Event Driven Executive messages and codes.

SCREEN TABLE LARGER THAN INPUT BUFFER: The screen table built during initialization exceeds the Input Buffer size.

Increase the Input Buffer size in module CDMCOMMN.

PROGRAM FILE LARGER THAN PROGRAM MANAGER BUFFER: The program table built during initialization exceeds the size of the buffer used by the program manager.

Increase the program manager buffer size in module CDMCOMMN.

TERMINAL TABLE OR WORK SPACE SIZE EXCEEDED: While building the terminal table and loading servers, the storage size or the the maximum number of terminals (10) allowed has been exceeded. The work space, defined in CDMINIT, is defined to allow a maximum of 50 terminals. The terminal table size can be increased by changing module CDMCOMMN.

BBBBBBBB SCREEN SIZE TOO LARGE: The specified screen in the SCRNS volume will not fit in the screen manager buffer.

Increase the screen manager buffer size in CDMCOMMN.

179

BBBBBBB SETPAN FAILED, RC=CCCCCC: A SETPAN failed for the screen name specified. Determine the cause of failure using the return code and the Multiple Terminal Manager SETPAN documentation.

**TERMINAL BBBBBBB BUSY:** A terminal specified in the TERMINAL file is connected to another program.

Try to RECONNECT at a later time.

ERROR ENCOUNTERED DURING CLOSE OF INDEXED ACCESS METHOD (DDDDDDDD, VVVVVV), ERROR CODE=(ccccc): An error occurred during AUTOCLOSE of an Indexed Access Method data set.

INITIALIZATION ERROR: Initialization has been unsuccessful. Multiple Terminal Manager is terminated. This message is written to the terminal which loaded Multiple Terminal Manager. Additional messages are printed on \$SYSPRTR.

INVALID PROGRAM NAME: The name of the program requested from the primary menu was not found in the Multiple Terminal Manager program table or invalid parameters supplied on a DISCONNECT command.

INVALID TERMINAL: The terminal name entered with a DISCONNECT command is not a Multiple Terminal Manager terminal.

**PROGRAM LOAD ERROR:** An Event Driven Executive LOAD error occurred for the requested program.

DISK READ ERROR: An internal Multiple Terminal Manager disk Read error has occurred and results may be unpredictable.

TERMINAL BBBBBBB RECONNECTED: The named terminal has been reconnected to the Multiple Terminal Manager.

**RECONNECT SYNTAX INVALID:** The RECONNECT operator interface facility is invalid and the proper syntax has not been used.

RECONNECT TERMINAL DEFINITION ERROR: The RECONNECT operator interface facility has encountered a failure while attempting to reconnect a terminal to the Multiple Terminal Manager. Since initialization would have already performed all functions necessary to include the terminal in the terminal table, the TERMINAL file, SCRNS volume or source table in RECONNEC has probably been altered since the Multiple Terminal Manager was started.

BBBBBBB DISCONNECT: Terminal bbbbbbbb has been issued a successful DISCONNECT command.

MULTIPLE TERMINAL MANAGER SYSTEM FAILURE: The Multiple Terminal Manager task error exit routine has been entered due to a machine or program error. The Multiple Terminal Manager program remains active waiting for an event which will not be posted.

The PSW and LSB at the time of failure has been saved at a displacement of X'172' into the program storage. Register 1 in the LSB contains the address of the failing instruction in the case of a program check. Use Event Driven Executive operator facilities to display storage.

An example follows showing a specification check which occurred at location X'053C'.

MULTIPLE TERMINAL MANAGER SYSTEM FAILURE > \$A

0172: **8002** 28E6 0110 10D0 0DDC **053C** 0DAC 7361 0182: 0540 815C 00B8 0DDA 0000 00FA 0004 0028 0192: 0052 007C 00A6 0017 0E72 A0A2 0E72 FFFF 01A2: 0102 8026 1616 40C9 D5C9 E3C9

ANOTHER DISPLAY?

The PSW is 8002 at 0172 and R1 is 053C on same line.

#### **EXAMPLE - FILE MAINTENANCE TRANSACTION APPLICATION**

This example consists of a pair of programs which perform a simple file maintenance task. The tasks it can perform are reading or writing a single record, or setting an end of data (EOD) marker. Both programs are presented in the following languages:

- Event Driven Language (see "EDL Sample Prog1" on page 190 and "EDL Sample Prog2" on page 191)
- COBOL (see "COBOL Sample Prog1" on page 193 and "COBOL Sample Prog2" on page 195)
- FORTRAN (see "FORTRAN Sample Prog1" on page 197 and "FORTRAN Sample Prog2" on page 198)
- PL/I (see "PL/I Sample Prog1" on page 200 and "PL/I Sample Prog2" on page 202)

The first program displays a screen which requests the file parameters which include data set name and relative record number. It then LINKs to the second program, passing the file parameters.

The second program builds a file control area (FCA) from the file parameters and performs the requested file I/O operation. The results of the operation are displayed on the screen, and the program ends.

The following is a detailed explanation of each program statement in Event Driven Language and the effects of program execution of the application.

The first statements in the first program are declarations.

EXTRN BEEP, SETPAN, MENU, ACTION, LINK
ENTRY MTMSUB
SUBROUT MTMSUB, INBADDR, OUTBADDR, TEBADDR, IIBADDR

EXTRN declares Multiple Terminal Manager functions as external, so they may be accessed by the application. ENTRY declares the application as an entry point. All Multiple Terminal Manager applications are subroutines, as depicted in the SUBROUT statement, called MTMSUB. They all have four parameters, the addresses of the Input Buffer, Output Buffer, Terminal Environment Block and Interrupt Information Byte. (The latter two are not used in this example.)

The next instructions put the buffer addresses into registers 1 and 2.

MOVE #1, INBADDR MOVE #2, OUTBADDR

The terminal is prepared to sound the audible alarm by:

CALL BEEP

A screen image is retrieved from a disk data set and placed into the buffers.

CALL SETPAN, (REQSCRN), (RC)

RC DATA F'0'
REQSCRN DATA CL8'REQ'

A screen image consists of two portions. These are protected data, which may be considered a screen template or form, and unprotected data, usually considered default information. The protected data is a screen sized (24 x 80) image consisting of character data which is displayed, and fields of nulls used for data entry. Default data is written by the ACTION call into these null fields and operator inputs are read from them. (Screen images are constructed using the \$IMAGE utility. See the Utilities, Operator Commands, Program Preparation, Mesages and Codes for detailed information on \$IMAGE.)

Note that both the protected and unprotected parts of a screen built by \$IMAGE must be explicitly initialized by the user; failure to do so causes CALL SETPAN to return return code 2 when the screen is retrieved for use by an application program.

After the call to SETPAN, the Input Buffer contains the screen as shown in SCREEN 1, with five null fields as depicted by dollar signs. The \$ is for illustrative purposes only, null fields are actually displayed as blanks.

#### SCREEN 1

 The Output Buffer contains data used to initialize (unprotected) input fields. It consists of 14 blanks, followed by READOOO1, followed by 80 blanks. When written to the unprotected portion of the screen, the terminal appears as shown in SCREEN 2. An example of SCREEN 2 is on the following pages.

(14 BLANKS) READOOO1

(80 BLANKS)

The Input Buffer holds the screen format, and the Output Buffer contains fields to initialize input fields.

A test of the return code from SETPAN is done. If the return code does not indicate a successful return, the program ends by giving control to the primary menu routine.

IF (RC,NE,-1) CALL MENU

ENDIF

Call the ACTION routine to display the contents of the buffers, and read the operator response.

CALL ACTION

ACTION's effects are:

- Write the Input Buffer's contents to the terminal as protected characters.
- Write the Output Buffer contents, if any, into the null fields as unprotected characters.
- Wait for the operator to enter data and press ENTER or a PF key.
- Read the contents of the unprotected fields, (that is the operator input) into the Input Buffer.

This results in SCREEN 2 appearing on the terminal, where the default characters are highlighted.

#### SCREEN 2

DATA SET, VOLUME NAME ==>
REQUEST (READ, WRIT, SEOD) ==>READ
RELATIVE RECORD NUMBER ==>0001
NUMBER OF RECORDS ==>1
DATA TO BE WRITTEN:

The operator then enters data, changing the default data associated with relative record number. For example, to read the third record of data set "K" on volume EDX013, the following data would be entered. See highlighted fields on SCREEN 3.

### SCREEN 3

DATA SET, VOLUME NAME ==>K ,EDX013

REQUEST (READ, WRIT, SEOD) ==>READ

RELATIVE RECORD NUMBER ==>0003

NUMBER OF RECORDS ==>1

DATA TO BE WRITTEN:

The operator signals that the input is ready by pressing ENTER or a PF key. ACTION then completes the input cycle by reading the contents of the unprotected fields into the Input Buffer. See the following example of the Input Buffer.

K

In order for PROG2 ("EDL Sample Prog2" on page 191, "COBOL Sample Prog2" on page 195, "FORTRAN Sample Prog2" on page 198 and "PL/I Sample Prog2" on page 202), to receive the file parameters they must be passed through the Output Buffer. The next instruction moves the input data from the Input Buffer to the Output Buffer.

MOVE (0,#2),(0,#1),(106,BYTES)

Finally, PROG2 is LINKed to.

CALL LINK, (IOPROG)

IOPROG DATA CL8'PROG2'

A call to MENU to terminate the transaction is placed after the LINK, in case the LINK is unsuccessful.

CALL MENU

The first four lines of PROG2 are similar to those of PROG1, except that other functions are declared external, and only register 2 is assigned a buffer address.

EXTRN FILEIO, SETPAN, MENU, ACTION
ENTRY MTMSUB
SUBROUT MTMSUB, INBADDR, OUTBADDR, TEBADDR, IIBADDR
MOVE #2, OUTBADDR

At this point the Output Buffer (pointed to by register #2) contains various file parameters. A file control area (FCA) is constructed using these parameters. For example, the request type is moved from the Output Buffer to the FCA.

MOVE FCAREQ, (REQTYPE, #2), (4, BYTES)

FCAREQ DATA CL4''

REQTYPE EQU 14

Similarly, other fields must be moved, and relative record number must be converted to numeric.

```
* SET UP FILE CONTROL AREA AND BUFFER.
         MOVE
                  FCAREQ, (REQTYPE, #2), (4, BYTES) REQUEST TYPE
         MOVE
                  FCADSN, (DSNAME, #2), (8, BYTES)
                                                   DATA SET NAME
         MOVE
                  FCANUM, 1
                                                   NUMBER OF RECS
                  FCARRN, (RRN, #2), FORMAT = (4,0,1)
                                                     CONVERT RRN
          CONVTD
         MOVE
                  FCAVOL, (VOLNAME, #2), (6, BYTES)
                                                     VOLUME NAME
                  BUFFER, (BUFFDISP, #2), (80, BYTES) DATA BUFFER
         MOVE
* FILE CONTROL AREA.
FCA
         EQU
                   CL4' '
FCAREQ
         DATA
                                      REQUEST TYPE
FCADSN
         DATA
                   CL8' '
                                      DATA SET NAME
                   F'1'
FCANUM
         DATA
                                      NUMBER OF RECORDS
                   FIOI
         DATA
                   F'0'
FCAEOD
         DATA
                                      EOD RELATIVE RECORD NUMBER
         DATA
                   Fini
FCARRN
         DATA
                   F'0'
                                      RELATIVE RECORD NUMBER
FCAVOL
                   CL6' '
                                      VOLUME NAME
         DATA
* EQUATES FOR OUTPUT BUFFER DATA.
DSNAME
         EQU
                   0
                                      DATA SET NAME
VOLNAME
         EQU
                   8
                                      VOLUME NAME
REQTYPE
         EQU
                   14
                                      REQUEST TYPE
                                      RELATIVE RECORD NUMBER
RRN
          EQU
                   18
BUFFDISP EQU
                   22
                                      BUFFER DISPLACEMENT
EODRRN
         EQU
                   102
                                      EOD RRN DISPLACEMENT
RCDISP
         EQU
                   106
                                      RETURN CODE DISPLACEMENT
```

A screen image with which to display the file data is retrieved, and the return code is checked. This screen is similar to the previous screens shown with the addition of two new fields.

CALL SETPAN, (LISTSCRN), (RC)
IF (RC, NE, -1)
CALL MENU

ENDIF

LISTSCRN DATA CL8'LST'

At this point the image depicted in SCREEN 4 is in the buffers. Since there is no default data, the Output Buffer is empty.

### SCREEN 4

DATA SET, VOLUME NAME ==>,
REQUEST (READ, WRIT, SEOD) ==>
RELATIVE RECORD NUMBER ==>
NUMBER OF RECORDS ==>1
DATA TO BE WRITTEN:

EOD RELATIVE RECORD NUMBER ==>
RETURN CODE ==>

The actual FILEIO operation is performed, specifying the FCA, a buffer, and a return code.

CALL FILEIO, (FCA), (BUFFER), (RC)

RC DATA F'0' BUFFER DATA 256X'0'

Note that the buffer is 256-bytes in length (the length of an Event Driven Executive record) even though only the first 80 bytes are used.

Now that all the file data is available, it is placed in the Output Buffer so that it can be displayed. The data is taken from the FCA, the buffer and return code, and concatenated so that it may be written into the unprotected fields of the screen image.

\* PUT DATA INTO OUTPUT BUFFER SO IT WILL BE DISPLAYED.

MOVE (REQTYPE, #2), FCAREQ, (4, BYTES) REQUEST TYPE MOVE (DSNAME,#2),FCADSN,(8,BYTES) DATA SET NAME CONVTB (EODRRN, #2), FCAEOD, FORMAT = (4,0,1) CONV EOD RRN CONVTB (RRN, #2), FCARRN, FORMAT = (4,0,I)CONVERT RRN MOVE (VOLNAME, #2), FCAVOL, (6, BYTES) **VOLUME NAME** MOVE (BUFFDISP,#2),BUFFER,(80,BYTES) DATA CONVTB (RCDISP, #2), RC, FORMAT=(4,0,1) CONV RET CODE The Output Buffer now looks as follows:

K EDX013READ0003RECORD 3(72 blanks)0005-001

Both Input and Output buffers are displayed on the screen by the following:

CALL ACTION

The following is an example of the displayed screen:

SCREEN 5

DATA SET, VOLUME NAME ==>K ,EDX013
REQUEST (READ, WRIT, SEOD) ==>READ
RELATIVE RECORD NUMBER ==>0003
NUMBER OF RECORDS ==>1
DATA TO BE WRITTEN:

RECORD 3

EOD RELATIVE RECORD NUMBER ==>0005 RETURN CODE ==>0000

A call to ACTION waits for operator input followed by an ENTER or PF key. In this case no input is desired; however, the use of ACTION allows the user to view the screen and press ENTER after the contents have been read. At that point the program ends.

CALL MENU

The following pages contain the applications used to perform the example previously shown.

The first sample application uses Event Driven Language, the second uses COBOL, the third FORTRAN, and the fourth PL/I.

#### EDL Sample Prog1

```
BEEP, SETPAN, MENU, ACTION, LINK
        EXTRN
        ENTRY
                MTMSUB
                MTMSUB, INBADDR, OUTBADDR, TEBADDR, IIBADDR
        SUBROUT
                #1, INBADDR
                                  GET INPUT BUFF ADDRESS
        MOVE
                                   GET OUTPUT BUFF ADDRESS
                #2,OUTBADDR
        MOVE
* BEEP UPON TERMINAL IO.
        CALL
                BEEP
* RETRIEVE SCREEN IMAGE AND ABORT IF ERROR.
                SETPAN, (REQSCRN), (RC) GET SCREEN IMAGE
        CALL
                 (RC, NE, -1)
                                    OK?
        ΙF
                   MENU
                                    NO
          CALL
        ENDIF
* DISPLAY SCREEN IMAGE, READ OPERATOR RESPONSE.
        CALL
                ACTION
* MOVE DATA FROM INPUT BUFFER TO OUTPUT BUFFER (106 BYTES).
        MOVE
                (0,#2),(0,#1),(106,BYTES)
* LINK TO PROGRAM WHICH WILL PERFORM FILE IO.
        CALL
                LINK, (IOPROG)
* ABORT IF LINK FAILS.
        CALL
                MENU
***********************
¥
                                                        ×
×
        DATA ITEMS
                                                        ×
                                                        ×
***********************
REQSCRN
        DATA
               CL8'REQ'
                                     NAME OF REQUEST SCREEN
                                     NAME OF IO PROGRAM
IOPROG
        DATA
               CL8'PROG2'
RC
        DATA
               F'0'
                                     RETURN CODE
        ENDPROG
        END
```

### EDL Sample Prog2

```
FILEIO, SETPAN, MENU, ACTION
         EXTRN
         ENTRY
                 MTMSUB
         SUBROUT MTMSUB, INBADDR, OUTBADDR, TEBADDR, IIBADDR
         MOVE
                 #2,OUTBADDR
                                          GET O/P BUFFER ADDR
* SET UP FILE CONTROL AREA AND BUFFER.
         MOVE
                 FCAREQ, (REQTYPE, #2), (4, BYTES) REQST TYPE
         MOVE
                 FCADSN, (DSNAME, #2), (8, BYTES)
                                                 DATA SET NAME
                                                NUMBER OF RECS
         MOVE
                 FCANUM, 1
         CONVTD
                 FCARRN, (RRN, #2), FORMAT = (4, 0, I)
                                                   CONVERT RRN
         MOVE
                 FCAVOL, (VOLNAME, #2), (6, BYTES)
                                                   VOLUME NAME
         MOVE
                 BUFFER, (BUFFDISP, #2), (80, BYTES) DATA BUFFER
* RETRIEVE LISTING SCREEN AND ABORT IF ERROR.
         CALL
                 SETPAN, (LISTSCRN), (RC)
         IF
                  (RC,NE,-1)
                                          GOT SCREEN IMAGE OK?
           CALL
                  MENU
                                          NO
         ENDIF
  PERFORM FILE IO.
                 FILEIO, (FCA), (BUFFER), (RC)
         CALL
* PUT DATA INTO OUTPUT BUFFER SO IT WILL BE DISPLAYED.
         MOVE
                 (REQTYPE, #2), FCAREQ, (4, BYTES) REQUEST TYPE
         MOVE
                  (DSNAME, #2), FCADSN, (8, BYTES)
                                                  DATA SET NAME
         CONVTB
                 (EODRRN, #2), FCAEOD, FORMAT = (4,0,I) CONV EOD RRN
                                                    CONVERT RRN
         CONVTB
                 (RRN, #2), FCARRN, FORMAT = (4,0,1)
         MOVE
                  (VOLNAME, #2), FCAVOL, (6, BYTES)
                                                    VOLUME NAME
         MOVE
                  (BUFFDISP, #2), BUFFER, (80, BYTES) DATA
         CONVTB
                 (RCDISP,#2),RC,FORMAT=(4,0,I)
                                                    CONV RET CODE
* DISPLAY SCREEN IMAGE AND DATA.
         CALL
                 ACTION
* END PROGRAM.
         CALL
                 MENU
*************************
¥
                                                              ¥
         DATA ITEMS
×
                                                              ×
                                                              ¥
***********************
                                    NAME OF LISTING SCREEN
LISTSCRN DATA
                  CL8'LST'
RC
         DATA
                  F'0'
                                    RETURN CODE
BUFFER
         DATA
                  256X'0'
                                    DATA BUFFER
* FILE CONTROL AREA.
FCA
         EQU
FCAREQ
         DATA
                  CL4' '
                                    REQUEST TYPE
FCADSN
         DATA
                  CL8' '
                                    DATA SET NAME
                  F'1'
FCANUM
                                    NUMBER OF RECORDS
         DATA
         DATA
                  F'0'
                  F'0'
FCAEOD
         DATA
                                    EOD RELATIVE RECORD NUMBER
                  F'0'
         DATA
FCARRN
         DATA
                  F'0'
                                    RELATIVE RECORD NUMBER
FCAVOL
         DATA
                  CL6' '
                                    VOLUME NAME
```

# EDL Sample Prog2 (continued)

* EQUATES	FOR OUT	UT BU	UFFER	DATA.	
DSNAME	EQU	0			DATA SET NAME
VOLNAME	EQU	8			VOLUME NAME
REQTYPE	EQU	14			REQUEST TYPE
RRN	EQU	18			RELATIVE RECORD NUMBER
BUFFDISP	EQU	22			BUFFER DISPLACEMENT
EODRRN	EQU	102			EOD RRN DISPLACEMENT
RCDISP	EQU	106			RETURN CODE DISPLACEMENT
	ENDPROG				
	END				

### COBOL Sample Prog1

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
   MTMSUB.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
    IBM-S1.
OBJECT-COMPUTER.
    IBM-S1.
DATA DIVISION.
WORKING-STORAGE SECTION.
77
    REQUEST-SCREEN
                    PIC X(8) VALUE "REQ
                                              ₩.
                      PIC X(8) VALUE "PROG2
77
    IO-PROGRAM
77
    RC
                      PIC S99 USAGE IS COMPUTATIONAL.
LINKAGE SECTION.
    INPUT-BUFFER.
    05 DATA-SET-NAME PIC X(8).
                      PIC X(6).
    05
        VOLUME-NAME
        REQUEST-TYPE PIC X(4).
    05
    05
        RELATIVE-RECORD-NUMBER PIC 9999.
    0.5
        BUFFER-DATA
                      PIC X(80).
01
    OUTPUT-BUFFER.
       DATA-SET-NAME PIC X(8).
    05
    05
       VOLUME-NAME
                      PIC X(6).
        REQUEST-TYPE PIC X(4).
    05
    05
        RELATIVE-RECORD-NUMBER PIC 9999.
    05 BUFFER-DATA
                      PIC X(80).
    05 EOD-RRN
                      PIC 9999.
    05 RETURN-CODE PIC 9999.
77
                    PIC X(100).
   TEB
                    PIC 99 COMP.
77
   IIB
```

# COBOL Sample Progl (continued)

¥

PROCEDURE DIVISION
USING INPUT-BUFFER, OUTPUT-BUFFER, TEB, IIB.
BEGIN.

\* BEEP UPON TERMINAL IO.

CALL "BEEP".

\* RETRIEVE SCREEN IMAGE AND ABORT IF ERROR.

CALL "SETPAN" USING REQUEST-SCREEN, RC.

IF RC IS NOT EQUAL TO -1

CALL "MENU".

\* DISPLAY SCREEN IMAGE, READ OPERATOR RESPONSE. CALL "ACTION".

\* MOVE DATA FROM INPUT BUFFER TO OUTPUT BUFFER.

MOVE CORRESPONDING INPUT-BUFFER TO OUTPUT-BUFFER.

\* LINK TO PROGRAM WHICH WILL PERFORM FILE IO. CALL "LINK" USING IO-PROGRAM.

\* ABORT IF LINK FAILS.
CALL "MENU".

RETURN-POINT. EXIT PROGRAM.

### COBOL Sample Prog2

```
IDENTIFICATION DIVISION.
PROGRAM-ID
    MTMSUB.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
    IBM-S1.
OBJECT-COMPUTER.
    IBM-S1.
DATA DIVISION.
WORKING-STORAGE SECTION.
                       PIC X(8) VALUE "LST
77
    LIST-SCREEN
                       PIC S99 USAGE IS COMP.
77
    RC
    BUFFER
                       PIC X(256).
77
    FILE-CONTROL-AREA.
                      PIC X(4).
    05
        REQUEST-TYPE
        DATA-SET-NAME PIC X(8).
    05
    05
        NUMBER-OF-RECORDS PIC S999 USAGE COMP VALUE 1.
                       PIC 599.
    05
        FILLER
                       PIC S999 USAGE IS COMP.
    05
        EOD-RRN
    05
        FILLER
                       PIC S99.
        RELATIVE-RECORD-NUMBER PIC S999 USAGE COMP.
    05
        VOLUME-NAME
                      PIC X(6).
LINKAGE SECTION.
01
    INPUT-BUFFER
                       PIC X(1920).
01
    OUTPUT-BUFFER.
    05
        DATA-SET-NAME PIC X(8).
    05
        VOLUME-NAME
                       PIC X(6).
    05
        REQUEST-TYPE
                       PIC X(4).
        RELATIVE-RECORD-NUMBER PIC 9999.
    05
                       PIC X(80).
    05
        BUFFER-DATA
                       PIC 9999.
    05
        EOD-RRN
        RETURN-CODE
                       PIC 9999.
    05
77
    TEB
                     PIC X(100).
77
                     PIC 99 COMP.
    IIB
```

195

# COBOL Sample Prog2 (continued)

×

PROCEDURE DIVISION
USING INPUT-BUFFER, OUTPUT-BUFFER, TEB, IIB.
BEGIN.

\* SET UP FILE CONTROL AREA.

MOVE CORRESPONDING OUTPUT-BUFFER TO FILE-CONTROL-AREA.

MOVE BUFFER-DATA TO BUFFER.

\* RETRIEVE LISTING SCREEN AND ABORT IF ERROR.

CALL "SETPAN" USING REQUEST-SCREEN, RC.

IF RC IS NOT EQUAL TO -1

CALL "MENU".

\* PERFORM FILE IO.

CALL "FILEIO" USING FILE-CONTROL-AREA, BUFFER, RC.

\* PUT DATA INTO OUTPUT BUFFER SO IT WILL BE DISPLAYED.

MOVE CORRESPONDING FILE-CONTROL-AREA TO OUTPUT-BUFFER.

MOVE BUFFER TO BUFFER-DATA OF OUTPUT-BUFFER.

MOVE RC TO RETURN-CODE OF OUTPUT-BUFFER.

\* DISPLAY SCREEN IMAGE.

CALL "ACTION".

\* END PROGRAM.

CALL "MENU".

RETURN-POINT.

EXIT PROGRAM.

### FORTRAN Sample Progl

```
*PROCESS NOCMPAT
      SUBROUTINE MTMSUB(INBUFF, OUTBUF, TEB, IIB)
      IMPLICIT INTEGER (A-Z)
      INTEGER TEB(50), IIB
      INTEGER*2 INBUFF(960), OUTBUF(512)
      EXTERNAL BEEP, SETPAN, ACTION, MENU, LINK
      REAL*8 REQSCR /'REQ '/, IOPROG /'PROG2 '/
      INTEGER RC
C
C BEEP UPON TERMINAL IO.
C
      CALL EDX(BEEP, 0)
C RETRIEVE SCREEN AND ABORT IF ERROR.
      CALL EDX(SETPAN, 2, IADDR(REQSCR), IADDR(RC) )
      IF (RC.NE.-1) CALL EDX(MENU, 0)
C
C DISPLAY SCREEN IMAGE, READ OPERATOR RESPONSE.
      CALL EDX(ACTION, 0)
C
C MOVE DATA FROM INPUT BUFF TO OUTPUT BUFF. (106 BYTES)
C
      DO 10 I=1,53
        OUTBUF(I) = INBUFF(I)
10
      CONTINUE
C
C LINK TO PROGRAM WHICH WILL PERFORM FILE IO.
      CALL EDX(LINK, 1, IADDR(IOPROG) )
C
C ABORT IF LINK FAILS.
      CALL EDX(MENU, 0)
      RETURN
      END
```

### FORTRAN Sample Prog2

```
*PROCESS NOCMPAT
      SUBROUTINE MTMSUB(INBUFF, OUTBUF, TEB, IIB)
      IMPLICIT INTEGER (A-Z)
      INTEGER TEB(50), IIB
      INTEGER*2 INBUFF(960), OUTBUF(512)
      EXTERNAL FILEIO, SETPAN, ACTION, MENU
      EXTERNAL $12COT,$12CIN
      INTEGER BUFFER(128)
      REAL*8 LSTSCR /'LST
      INTEGER RC, FOUR/4/, RES/0/
C FILE CONTROL AREAS
      INTEGER FCA(14)
C REQUEST TYPE
      EQUIVALENCE (REQ, FCA(1)), (REQ1, FCA(1)), (REQ2, FCA(2))
      INTEGER*4 REQ
      INTEGER*2 REQ1, REQ2
C DATA SET NAMES
      EQUIVALENCE (DSN, FCA(3))
      INTEGER DSN(4)
C NUMBER OF RECORDS
      EQUIVALENCE (NUMREC, FCA(7))
      INTEGER NUMREC /1/
C END OF DATA RELATIVE RECORD NUMBER
      EQUIVALENCE (EODRRN, FCA(9))
      INTEGER EODRRN
C RELATIVE RECORD NUMBER
      EQUIVALENCE (RRN, FCA(11))
      INTEGER RNN
C VOLUME NAME
      EQUIVALENCE (VOL, FCA(12))
      INTEGER VOL (3)
      CALL EDX(ACTION, 0)
C SET UP FILE CONTROL AREA.
C
      DO 10 I=1,4
        DSN(I) = OUTBUF(I)
10
      DO 20 I=1,3
20
        VOL(I) = OUTBUF(I+4)
      REQ1 = OUTBUF(8)
      REQ2 = OUTBUF(9)
C CONVERT RELATIVE RECORD NUMBER TO NUMERIC
      CALL $12CIN(RRN, FOUR, OUTBUF(10), RES, RES, RES, RES)
      DO 30 I=1.40
        BUFFER(I) = OUTBUF(I+11)
```

# FORTRAN Sample Prog2 (continued)

```
30
      CONTINUE
C
C RETRIEVE LISTING SCREEN AND ABORT IF ERROR.
     CALL EDX(SETPAN, 2, IADDR(LSTSCR), IADDR(RC) )
     IF (RC.NE.-1) CALL EDX(MENU, 0)
C
C PERFORM FILE IO.
C
     CALL EDX(FILEIO, 3, IADDR(FCA), IADDR(BUFFER), IADDR(RC))
C
C PUT DATA INTO OUTPUT BUFFER SO THAT IT IS DISPLAYED.
     DO 40 I=1,4
       OUTBUF(I) = DSN(I)
40
     CONTINUE
     DO 50 I=1,3
       OUTBUF(I+4) = VOL(I)
50
     CONTINUE
     OUTBUF(8) = REQ1
     OUTBUF(9) = REQ2
C
C CONVERT RELATIVE RECORD NUMBER TO EBCDIC
C
     CALL $12COT(RRN,FOUR,OUTBUF(10),RES,RES,RES,RES)
     DO 60 I=1,40
       OUTBUF(I+11) = BUFFER(I)
60
     CONTINUE
C
C CONVERT EOD RELATIVE RECORD NUMBER TO EBCDIC
     CALL $12COT(EODRRN, FOUR, OUTBUF(52), RES, RES, RES, RES)
C
C CONVERT RETURN CODE TO EBCDIC
C
     CALL $12COT(RC, FOUR, OUTBUF(54), RES, RES, RES, RES)
C
C DISPLAY SCREEN IMAGE.
C
      CALL EDX(ACTION, 0)
C END PROGRAM.
      CALL EDX(MENU, 0)
      RETURN
      END
```

### | PL/I Sample Prog1

```
MTMSUB: PROCEDURE (INPUT_BUFFER,
                    OUTPUT_BUFFER,
                    TEB,
                    IIB);
DECLARE
  01 INPUT_BUFFER,
    05 DATA_SET_NAME
                               CHARACTER (8),
                               CHARACTER (6),
    05 VOLUME_NAME
                               CHARACTER (4),
    05 REQUEST_TYPE
    05
        RELATIVE_RECORD_NUMBER CHARACTER (4),
    05
        BUFFER_DATA
                               CHARACTER (80);
DECLARE
  01 OUTPUT_BUFFER,
       DATA_SET_NAME
                               CHARACTER (8),
    05
    05
       VOLUME_NAME
                               CHARACTER (6),
    05 REQUEST_TYPE
                               CHARACTER (4),
    05 RELATIVE_RECORD_NUMBER PICTURE '9999',
        BUFFER_DATA
                               CHARACTER (80),
    05
        EOD_RRN
                               PICTURE '9999',
    05
    05
        RETURN_CODE
                               PICTURE 'S999';
DECLARE
  (TEB, IIB) BINARY FIXED (15);
DECLARE
  (SETPAN, ACTION, BEEP, LINK, MENU) ENTRY;
DECLARE
  REQUEST_SCREEN CHARACTER (8) INITIAL ('REQ') STATIC;
DECLARE
  PROGRAM_NAME CHARACTER (8) INITIAL ('PROG2') STATIC;
DECLARE
  RETURN_CODE BINARY FIXED (15) STATIC;
```

```
PL/I Sample Prog1 (continued)
     /* BEEP UPON TERMINAL IO. */
       CALL BEEP;
     /* RETRIEVE SCREEN IMAGE AND ABORT IF ERROR. */
       CALL SETPAN (REQUEST_SCREEN, RETURN_CODE);
       IF RETURN_CODE -= -1
         THEN CALL MENU;
     /* DISPLAY SCREEN IMAGE, READ OPERATOR RESPONSE. */
       CALL ACTION;
     /* MOVE DATA FROM INPUT BUFFER TO OUTPUT BUFFER */
       OUTPUT_BUFFER.DATA_SET_NAME = INPUT_BUFFER.DATA_SET_NAME;
       OUTPUT_BUFFER.VOLUME_NAME = INPUT_BUFFER.VOLUME_NAME;
       OUTPUT_BUFFER.REQUEST_TYPE = INPUT_BUFFER.REQUEST_TYPE;
       OUTPUT_BUFFER.RELATIVE_RECORD_NUMBER
         = INPUT_BUFFER.RELATIVE_RECORD_NUMBER;
       OUTPUT_BUFFER.BUFFER_DATA = INPUT_BUFFER.BUFFER_DATA;
     /* LINK TO PROGRAM WHICH WILL PERFORM FILE IO. */
       CALL LINK (PROGRAM_NAME);
     /* ABORT IF LINK FAILS. */
       CALL MENU;
     END;
```

# | PL/I Sample Prog2

```
MTMSUB: PROCEDURE (INPUT_BUFFER,
                    OUTPUT_BUFFER,
                    TEB,
                    PF_KEY);
DECLARE
  01 OUTPUT_BUFFER,
                           CHARACTER (8),
    05 DATA_SET_NAME
    05 VOLUME_NAME
                              CHARACTER (6),
    05 REQUEST_TYPE
                               CHARACTER (4),
    05 RELATIVE_RECORD_NUMBER PICTURE '9999',
    O5 BUFFER_DATA CHARACTER (80),
    05 EOD_RRN
                               PICTURE '9999',
                              PICTURE 'S999';
    05
        RETURN_CODE
DECLARE
  (INPUT_BUFFER, TEB, PF_KEY)
  BINARY FIXED (15);
DECLARE
  (SETPAN, ACTION, FILEIO, MENU) ENTRY;
DECLARE
  RETURN_CODE BINARY FIXED (15) STATIC;
DECLARE
  01 BUFFER
                               STATIC,
                               CHARACTER (80),
   05 FIRST_80
    05 LAST_176
                               CHARACTER (176);
DECLARE
  LIST_SCREEN CHARACTER (8) INITIAL ('LST') STATIC;
DECLARE
  01 FILE_CONTROL_AREA
                              STATIC,
    05
       REQUEST_TYPE
                               CHARACTER (4),
       DATA_SET_NAME
       DATA_SET_NAME CHARACTER (8),
NUMBER_OF_RECORDS BINARY FIXED (15) INITIAL (1),
    05
    05
    05
       FILLER1
                               BINARY FIXED (15),
                               BINARY FIXED (15),
    05
       EOD_RRN
    05
       FILLER2
                               BINARY FIXED (15),
        RELATIVE_RECORD_NUMBER BINARY FIXED (15),
    05
       VOLUME_NAME
    05
                               CHARACTER (6);
```

# | PL/I Sample Prog2 (continued)

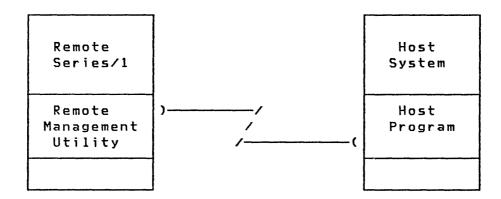
```
/* SET UP FILE CONTROL AREA. */
  FILE_CONTROL_AREA.REQUEST_TYPE =
    OUTPUT_BUFFER.REQUEST_TYPE;
  FILE_CONTROL_AREA.DATA_SET_NAME =
    OUTPUT_BUFFER.DATA_SET_NAME;
  FILE_CONTROL_AREA.VOLUME_NAME =
    OUTPUT_BUFFER.VOLUME_NAME;
  FILE_CONTROL_AREA.RELATIVE_RECORD_NUMBER =
    OUTPUT_BUFFER.RELATIVE_RECORD_NUMBER;
  BUFFER.FIRST_80 = OUTPUT_BUFFER.BUFFER_DATA;
/* RETRIEVE LISTING SCREEN AND ABORT IF ERROR. */
  CALL SETPAN (LIST_SCREEN, RETURN_CODE);
  IF RETURN_CODE -= -1
    THEN CALL MENU;
/* PERFORM FILE IO. */
  CALL FILEIO (FILE_CONTROL_AREA, BUFFER, RETURN_CODE);
/* MOVE DATA TO OUTPUT BUFFER SO IT WILL BE DISPLAYED. */
  OUTPUT_BUFFER.DATA_SET_NAME =
    FILE_CONTROL_AREA.DATA_SET_NAME;
  OUTPUT_BUFFER.VOLUME_NAME =
    FILE_CONTROL_AREA.VOLUME_NAME;
  OUTPUT_BUFFER.REQUEST_TYPE =
    FILE_CONTROL_AREA.REQUEST_TYPE;
  OUTPUT_BUFFER.RELATIVE_RECORD_NUMBER
    = FILE_CONTROL_AREA.RELATIVE_RECORD_NUMBER;
  OUTPUT_BUFFER.BUFFER_DATA = BUFFER.FIRST_80;
  OUTPUT_BUFFER.EOD_RRN = FILE_CONTROL_AREA.EOD_RRN;
  OUTPUT_BUFFER.RETURN_CODE = RETURN_CODE;
/* DISPLAY SCREEN IMAGE. */
  CALL ACTION;
/* END PROGRAM. */
  CALL MENU;
END;
```

The Event Driven Executive Remote Management Utility provides facilities for the management of a remote Series/1. The remote Series/1 is controlled by a host system. The utility waits for a request sent from the host, and then performs the particular function as specified by the request. Through implementation of this utility, the concept of distributed processing can be realized.

This chapter describes these facilities and their operation, discusses the interface requirements, and provides information about the installation and execution of the Remote Management Utility.

The Remote Management Utility runs as a program in the remote Series/1 and supports such functions as file allocation and transfer, and remote operator interaction, thus minimizing the need for an operator at the remote Series/1.

The remote Series/1 is controlled by the host system via a point-to-point or multipoint binary synchronous communication line using the Event Driven Executive Binary Synchronous Communication Access Method (BSCAM).



A user-written host program communicates with the Remote Management Utility via a record exchange. Through this record exchange, the host requests function execution on the remote system. Any system supporting BSCAM-compatible binary synchronous line protocol including transparency mode, and the Remote Management Utility record exchange interface may serve as the host system.

#### REMOTE MANAGEMENT FUNCTIONS

The utility provides various remote management functions that can be invoked through a request issued by the host program. Listed here is a brief description of the functions provided by the utility:

ALLOCATE Allocate a disk/diskette data set on the Series/1

DELETE Delete a disk/diskette data set on the Series/1

DUMP Dump storage to a disk/diskette data set on the Series/1

EXEC Initiate execution of a program on the Series/1

IDCHECK Verify identification between the host and the

Remote Management Utility

PASSTHRU Establish an interactive connection between the host and an application or utility on the remote Series/1

RECEIVE Receive data from the host and write it to an existing disk/diskette data set on the Series/1

SEND Read a disk/diskette data set on the Series/1 and transmit it to the host

SHUTDOWN Terminate the Remote Management Utility and free up any allocated resources; may also initiate execution of another program

WRAP Transmit a block of data just received back to the host

The section "Remote Management Utility Functional Operation" on page 213 describes in detail these functions and how they operate.

#### HARDWARE REQUIREMENTS

The Remote Management Utility requires approximately 7K bytes of storage plus buffer space. The default buffer space is 1024 bytes. In addition, the following are the minimum requirements:

- 4952, 4953, or 4955 processor (64K minimum recommended)
- One of the following BSC features:
  - Single-line adapter (#2074 or #2075)
  - Multiline controller (#2093) and one or two ,4-line adapters (#2094)
- Point-to-point (leased or switched) or multipoint (remote Series/1 as a tributary) binary synchronous communications line
- Disk or diskette
  - Disk (4962 or 4963)
  - Diskette (4964 or 4966)

### SOFTWARE REQUIREMENTS

The Remote Management Utility executes with Event Driven Executive Version 2.0. The Event Driven Executive utilities are required for the installation of the Remote Management Utility. A user-written program is required on the host to communicate with the Remote Management Utility.

## REMOTE MANAGEMENT UTILITY INTERFACE

The Remote Management Utility requires a user-written host program that will provide inter-program communication between the host system and the remote Series/1. The Remote Management Utility interface is comprised of two levels of communication: the binary synchronous communication (BSC) protocol, and a Remote Management Utility record exchange between the host system and the remote Series/1. A feature of the record exchange interface provides data-record blocking operations.

207

#### Binary Synchronous Communication Protocol

The Remote Management Utility uses the BSC protocol as defined by the Event Driven Executive BSCAM. A general introduction to binary synchronous communications and details of the line protocol can be found in <u>General Information — Binary Synchronous Communications</u>, GA27-3004. Specific implementations of BSC with the Remote Management Utility are as follows:

- The utility sends EOT as "abort". The host program should also send EOT to abort.
- The utility will not time out when receiving data. The host program may send TTD, which will be responded to by NAK.
- EOT is sent whenever the utility expects a delay. The utility will not send TTD in the event of unforeseen delays.
- Transparent EBCDIC mode is used exclusively. The host must be capable of communicating with transparent EBCDIC.
- Point-to-point communications (leased or switched) or multipoint communications are supported. If multipoint communications are used, the utility functions as a tributary on the multipoint line.

### Record Exchange

The second level of communication of the Remote Management Utility interface is that of a record exchange between the host and the remote Series/1.

Records are transmitted between the host system and the remote Series/1 in a predefined format. As the content of the record determines the function to be performed, this predefined format ensures that all necessary information is properly communicated between the host system and the remote. The host is responsible for formatting records sent to the remote Series/1, and processing records received from the remote Series/1. After receiving a function request, the utility sends a record containing a status code to the host signaling the result of the function execution.

#### Record Format

Each Remote Management Utility record has 4 bytes at the beginning, that are referred to as the header. The first 2 bytes of the header contain the BSC control characters DLE STX, and are represented as X'1002'. The third byte contains the character 'X', identifying it as an Event Driven Executive Remote Management Utility record. The fourth byte contains a character code identifying the record type. Figure 13 lists the various record types. The remainder of the record, or the record extension, is determined by the record type as specified in the header. There are 10 types of record extensions for a Request type record. Figure 14 on page 210 illustrates the structure of the Remote Management Utility record scheme.

The section at the end of this chapter, "CDRRM Equate Listing" on page 292 illustrates the various record types, including the extensions. This set of equates defining the Remote Management Utility record is obtainable through copy code "COPY CDRRM".

Code	Туре	Usage
R	Request	Sent by host to request a func- tion
S	Status	Sent by either system to indi- cate success or failure of a function
С	Count	Sent by the remote Series/1 after transfer of a data set, to indicate the number of data records processed
D	Data	Used for transfer of a data set
Р	Passthru	Used to pass data and data requests between the host and an application on the remote Series/1

Figure 13. Remote Management Utility Record Types

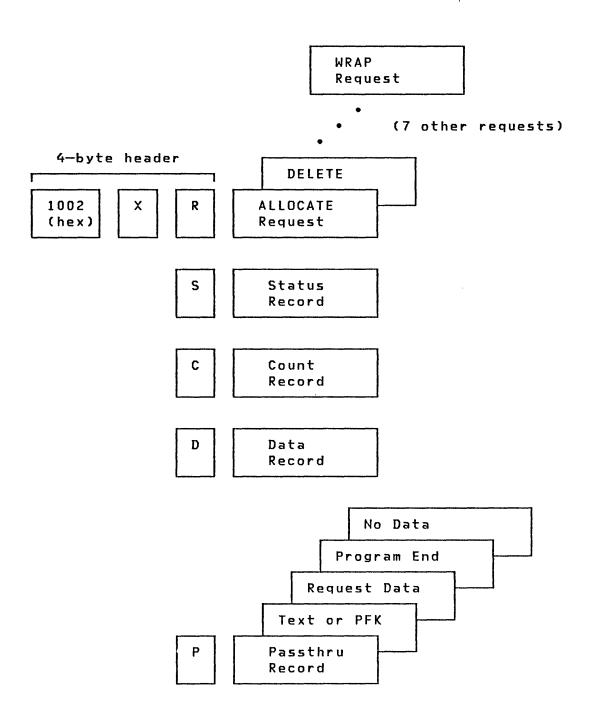


Figure 14. Remote Management Utility Record Scheme

## Record Blocking

On data transfer operations (SEND and RECEIVE), the Remote Management Utility performs two types of record blocking, which are performed independently of one another, and thus, may be combined. A field in the SEND and RECEIVE record header dynamically determines the number of 80-byte or 256-byte records to be sent over the BSC line per transmission. In addition, if data sets are specified as containing 80-byte records (as in Event Driven Executive source files), the redundant 48 bytes per line of text are not transmitted.

The following example illustrates a 256-byte record containing "text":

80 bytes	48 bytes	80 bytes	48 bytes
TEXT	(unused)	TEXT	(unused)

The use of blocking will increase the efficiency with which the communications line is used. This is for two reasons:

- Blocking decreases the amount of data transmitted. The 4-byte header, along with other communications control information is sent only once per block.
- Blocking decreases the number of delays associated with each message sent over a communications line.

Provided sufficient storage resources are available, it is advantageous to use large block sizes. However, the point occurs when, due to errors on the communications line, error recovery makes use of large blocks less efficient.

# Buffer Allocation

The Remote Management Utility contains a constant that determines the amount of storage to allocate for buffers. Records received by the utility may not exceed this buffer length. If a record is received greater than this length, a Status record indicating this condition (BSC I/O failure) is sent to the host and the function in progress (if any) is terminated. The default buffer size is 1K (1024 bytes). The section "Modifying Defaults" on page 283 describes how this buffer size may be modified.

### Parameter Passing

The EXEC, PASSTHRU, and SHUTDOWN functions of the Remote Management Utility allow programs to be loaded for execution if specified on the request. Many programs require parameters to be passed to them in the form of a character field. An example of some of the programs requiring parameters are \$EDXASM, \$LINK, and \$UPDATE, any of which may be specified on the request. The format of the parameter(s) to be passed is described in program preparation via the \$JOBUTIL utility in Utilities, Operator Commands, Program Preparation, Messages and Codes.

The parameter is coded for \$JOBUTIL on the PARM statement in columns 10 through 72. To provide the equivalent information on the PASSTHRU request for example, you should code a parameter of 64 characters with the same content as columns 10 through 72 of the PARM statement. The length of the parameter is 32 words.

The following two examples illustrate how parameters would be passed to \$EDXASM by way of the \$JOBUTIL utility and the Remote Management Utility via a PASSTHRU request:

#### \$JOBUTIL statements:

PROGRAM	\$EDXASM,ASMLIB
PARM	ERRORS *
DS	MYSRC, MYVOL
DS	ASMWORK
DS	ASMOBJ
EXEC	

### PASSTHRU Request:

```
RMHBSCC
          DATA
                 X'1002'
                 C'X'
RMHID
          DATA
                 C'R'
RMHTYP
          DATA
RMREQ
          DATA
                 F'12'
                 H'0'
          DATA
RMPRPTN
          DATA
                 H'0'
RMPRPGM
          DATA
                 CL8'$EDXASM'
RMPRVOL
          DATA
                 CL6'ASMLIB'
RMPRLFS
          DATA
                 F'0'
RMPRBLK
          DATA
                 F'0'
RMPRPRM#
          DATA
                 F'32'
RMPRPRM
          DATA
                CL64'ERRORS
                                  * 1
RMPRDS#
          DATA
                 F'3'
RMPRDS
          DATA
                 CL14'MYSRC
                               MYVOL'
          DATA
                CL14'ASMWORK'
          DATA CL14'ASMOBJ'
```

### REMOTE MANAGEMENT UTILITY FUNCTIONAL OPERATION

This section describes the remote management functions in detail, including the communications flow and record formats for each function. The section "Sample Host Programs" on page 259 illustrates several host programs which perform some of the functions provided by the Remote Management Utility.

The examples in this section of the communications flow between the host and the remote Series/1 reflect the BSCAM level of access used by the host program and the utility. The DATA statements in these examples reflect code passed to the utility from the host program. The responses sent to the host from the utility are preceded by equal signs (=). Additional detail on the access method and BSC functions can be found in "Chapter 3. Binary Synchronous Communications" on page 35.

#### ALLOCATE Function

The ALLOCATE function requests the utility to allocate a disk/diskette data set on the remote Series/1.

The host sends the remote Series/1 a Request record with the ALLOCATE function specified. After receiving and executing the ALLOCATE request, the utility sends a Status record to the host indicating the results of the function execution. The utility then waits for a new request from the host.

The ALLOCATE function uses the \$DISKUT3 utility in performing its function. Thus, data sets with the names: \$EDXNUC, \$\$EDXVOL, and \$\$EDXLIB may not be allocated with the ALLOCATE function.

# Required Field Descriptions

Specify the following fields for the ALLOCATE function:

- RMHBSCC A 2-byte hexadecimal field containing the BSC control characters DLE STX, specified as X'1002'.
- RMHID A 1-byte alphameric field containing the header ID 'X', identifying the record as an Event Driven Executive Remote Management Utility record.
- RMHTYP A 1-byte alphameric field identifying the header type. This field contains the character 'R', specifying a Request record type.
- RMREQ A 2-byte numeric field specifying the request type. For an ALLOCATE request, this field contains the number 2.
- RMADSN An 8-byte alphameric field containing the name of the data set to be allocated.
- RMAVOL A 6-byte alphameric field specifying the name of the volume on which the data set is to be allocated. If RMAVOL is blank, the volume name defaults to the IPL volume.
- RMANREC A 4-byte (double word) numeric field containing the number of 256-byte records to be allocated for the data set. Only the second word of this field is used.

A 2-byte numeric field identifying the type of data set to be allocated. Specify one of the following types:

- O User defined
- 1 Data
- 3 Program

Figure 15 illustrates the host-remote interaction for the ALLOCATE function. In the example, the host requests a data set named "MYDATA" to be allocated on the volume "MYVOL". The data set type is 1 (data) and is to contain ten 256-byte records. The remote sends a status of -1 (successful) to the host, and the operation is completed.

<u>Host Program</u>	<u>Host</u>		Remote
Write Initial - Request	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'			
		<	ACK*
Write End	EOT	>	
Read Initial - Status		<	ENQ
	ACK*	>	
RMHTYP='S'		<	TEXT
RMSREQ=2			
RMSFN=-1			
Read Continue - EOT	ACK*	>	
	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	<	FOT

Figure 15. Communications Flow for the ALLOCATE Function

### **DELETE Function**

The DELETE function requests the utility to delete a disk/diskette data set on the remote Series/1.

The host sends the remote Series/1 a Request record with the DELETE function specified. After receiving and executing the DELETE request, the utility sends a Status record to the host indicating the results of the function execution. The utility then waits for a new request from the host.

The DELETE function uses the \$DISKUT3 utility in performing its function. Thus, data sets with the names: \$EDXNUC, \$\$EDXVOL, and \$\$EDXLIB may not be deleted with the DELETE function.

## Required Field Descriptions

Specify the following fields for the DELETE function:

- RMHBSCC A 2-byte hexadecimal field containing the BSC control characters DLE STX, specified as X'1002'.
- RMHID A 1-byte alphameric field containing the header ID 'X', identifying the record as an Event Driven Executive Remote Management Utility record.
- RMHTYP A 1-byte alphameric field identifying the header type. This field contains the character 'R', specifying a Request record type.
- RMREQ A 2-byte numeric field specifying the request type. For a DELETE request, this field contains the number 3.
- RMDDSN An 8-byte alphameric field containing the name of the data set to be deleted.
- RMDVOL A 6-byte alphameric field specifying the name of the volume that contains the data set to be deleted. If RMDVOL is blank, the volume name defaults to the IPL volume.

Figure 16 illustrates the host-remote interaction for the DELETE function. In the example, the host specifies a data set named "MYDATA" to be deleted from the volume "MYVOL". The remote sends a status of -1 (successful) to the host, and the operation is completed.

		<del></del>	
<u>Host Program</u>	<u>Host</u>		Remote
Write Initial - Request	ENQ	>	VCK*
RMHBSCC DATA X'1002' RMHID DATA C'X' RMHTYP DATA C'R' RMREQ DATA F'3'	TEXT	>	
RMDDSN DATA CL8'MYDATA' RMDVOL DATA CL6'MYVOL'			
Write End	EOT	<>	ACK*
Read Initial - Status	ACK*	<>	
RMHTYP='S' RMSREQ=3 RMSFN=-1		<	TEXT
Read Continue – EOT	ACK*	> <	ЕОТ

Figure 16. Communications Flow for the DELETE Function

### **DUMP** Function

The DUMP function requests the utility to dump an Event Driven Executive storage partition to a disk/diskette data set on the remote Series/1.

The host sends the remote Series/1 a Request record with the DUMP function specified. After receiving and executing the DUMP request, the utility sends a Status record to the host indicating the results of the function execution. The utility then waits for a new request from the host.

## Required Field Descriptions

Specify the following fields for the DUMP function:

- RMHBSCC A 2-byte hexadecimal field containing the BSC control characters DLE STX, specified as X'1002'.
- RMHID A 1-byte alphameric field containing the header ID 'X', identifying the record as an Event Driven Executive Remote Management Utility record.
- RMHTYP A 1-byte alphameric field identifying the header type. This field contains the character 'R', specifying a Request record type.
- RMREQ A 2-byte numeric field specifying the request type. For a DUMP request, this field contains the number 4.
- RMDPDSN An 8-byte alphameric field containing the name of a previously allocated data set into which the storage of the partition is to be dumped.
- RMDPVOL A 6-byte alphameric field specifying the name of the volume containing the dump data set. If RMDPVOL is blank, the volume name defaults to the IPL volume.
- filler A 1-byte reserved field (unused).
- RMDPPTN A 1-byte numeric field specifying the partition to be dumped. Specify one of the following:
  - -1 Remote Management Utility partition 1-8 Specific partition

Figure 17 illustrates the host-remote interaction for the DUMP function. In the example, the host requests that partition 1 be dumped to the data set "MYDATA" on the volume "MYVOL". The remote sends a status of -1 (successful) to the host, and the operation is completed.

<u>Host Program</u>	<u>Host</u>		Remote
Write Initial - Request	ENQ	>	
RMHBSCC DATA X'1002'	TEVT	<>	ACK*
RMHID DATA C'X'	IEXI		
RMHTYP DATA C'R'			
RMREQ DATA F'4' RMDPDSN DATA CL8'MYDATA'			
RMDPVOL DATA CL6'MYVOL'			
DATA H'O'			
RMDPPTN DATA H'1'		<	4 C V X
Write End	EOT		ACK*
Read Initial - Status		<	ENQ
	ACK*	>	
RMHTYP='S' RMSREQ=4		<	TEXT
RMSFN=-1			
Daniel Cambinus 507	4 G V v		
Read Continue - EOT	AUK*	> <	EOT
		-	- ·

Figure 17. Communications Flow for the DUMP Function

#### **EXEC** Function

The EXEC function requests the utility to load and invoke execution of a program on the remote Series/1.

The hosts sends the remote Series/1 a Request record with the EXEC function specified. After receiving and executing the EXEC request, the utility sends a Status record to the host indicating the results of the function execution. The utility then waits for a new request from the host.

If the program specified by the host requires a parameter and the parameter is not supplied, the load (via LOAD) of the program will fail. For further information on parameter passing, refer to the section "Parameter Passing" on page 212.

## Required Field Descriptions

Specify the following fields for the EXEC function:

- RMHBSCC A 2-byte hexadecimal field containing the BSC control characters DLE STX, specified as X'1002'.
- RMHID A 1-byte alphameric field containing the header ID 'X', identifying the record as an Event Driven Executive Remote Management Utility record.
- RMHTYP A 1-byte alphameric field identifying the header type. This field contains the character 'R', specifying a Request record type.
- RMREQ A 2-byte numeric field specifying the request type. For an EXEC request, this field contains the number 9.
- filler A 2-byte reserved field (unused).
- RMXFLG A 1-byte numeric field containing the RMXFLGL and RMXFLGW bits. RMXFLGL and RMXFLGW correspond to the usage of the LOGMSG and WAIT parameters of the Event Driven Language LOAD instruction.

RMXFLGL - When set on, this bit indicates that a "program loaded" message is to be printed on the terminal which loaded the utility. The value for RMXFLGL when set on is X'40'.

RMXFLGW - When set on, this bit indicates that the utility is to wait for the completion of the program before sending a Status record to the host. Otherwise, the program executes asynchronously with the utility, and the utility sends a Status record after invoking the LOAD instruction. If the utility waits for the completion of the program, the PROGSTOP code from the program is returned in the RMSST field of the Status record. The value for RMXFLGW when set on is X'20'.

RMXPTN A 1-byte numeric field specifying the partition the program is to run in. Specify one of the following:

- -1 Remote Management Utility partition
  - 0 Any partition
- 1-8 Specific partition

RMXPGM An 8-byte alphameric field specifying the program to be executed.

RMXVOL A 6-byte alphameric field specifying the name of the volume which contains the program. If RMXVOL is blank, the volume name defaults to the IPL volume.

RMXLFS A 2-byte numeric field specifying the amount of free space (in bytes) to pass to the program.

RMXPRM# A 2-byte numeric field specifying the length of the parameter(s), in words, to pass to the program. This field must be zero if no parameters are passed.

RMXPRM A variable length field containing the parameter(s) to be passed to the program. The length of this field, in words, must correspond to the value contained in the RMXPRM# field. See the section "Parameter Passing" on page 212 for details on this field.

RMXDS# A 2-byte numeric field specifying the number of data set names to pass to program. The maximum number of data sets that may be specified is nine. This field must be zero if no data set names are passed.

RMXDS A variable number of 14-byte alphameric fields specifying the data set and volume names to be passed to the program. The first eight bytes contain the data set name, and the last six bytes contain the volume name. If the volume name is blank, the name of the volume defaults to the IPL volume. The number of data set and volume names specified must correspond to the value contained in the RMXDS# field.

Figure 18 illustrates the host-remote interaction for the EXEC function. In the example, the host specifies a program named "MYPROG" on the 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 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. The remote sends a status of -1 (successful) to the host, along with the return code and the operation is completed.

<u>Host Program</u>		<u>Host</u>		<u>Remote</u>	
Write In	nitial	- Request	ENQ	> <	A C K ¥
RMHBSCC	DATA	X'1002'	TEXT	·>	AOK^
RMHID	DATA				
RMHTYP					
RMREQ	DATA	F'9'			
•	DATA	F'0'			
RMXFLG	DATA	X'60'			
RMXPTN	DATA	H'1'			
RMXPGM		CL8'MYPROG'			
RMXVOL	DATA	CL6'MYVOL'			
RMXLFS	DATA	F'256'			
RMXPRM#	DATA	F'0'			
RMXPRM	EQU	*			
RMSDS#	DATA	F'0'			
RMSDS	EQU	*			
				<	ACK×
Write Er	nd		EOT	>	
Read Ini	itial -	Status		<	ENQ
			ACK×	>	
RMHTYP='	'S'			<	TEXT
RMSREQ=9	)				
RMSFN=-1					
RMSST=-1	L				
Read Cor	ntinue	- EOT	ACK*	>	
				<	EOT

Figure 18. Communications Flow for the EXEC Function

#### IDCHECK Function

The IDCHECK function allows the host and the remote system to verify each others identification.

The host sends the remote Series/1 a Request record with the IDCHECK function and the host ID specified. The utility compares this ID with a constant defined in the utility as the host ID. If the IDs match, the utility returns a Status record which contains the ID of the remote system, which is another constant. If the IDs do not match, an error status is returned to the host and the ID of the remote Series/1 is not returned. In either case, after the Status record is sent to the host, the utility then waits for a new request from the host.

The default host ID for the host system is "HOSTRMUX", and "REMTRMUX" is the default ID of the remote system.

# Required Field Descriptions

Specify the following fields for the IDCHECK function:

- RMHBSCC A 2-byte hexadecimal field containing the BSC control characters DLE STX, specified as X'1002'.
- RMHID A 1-byte alphameric field containing the header ID 'X', identifying the record as an Event Driven Executive Remote Management Utility record.
- RMHTYP A 1-byte alphameric field identifying the header type. This field contains the character 'R', specifying a Request record type.
- RMREQ A 2-byte numeric field specifying the request type. For an IDCHECK request, this field contains the number 6.
- RMICHK An 8-byte alphameric field specifying the host ID.

Figure 19 illustrates the host-remote interaction for the IDCHECK function. In the example, the host specifies the ID "HOSTRMUX". The remote validates the host ID, sends a status of -1 (successful) to the host along with the remote system's ID, "REMTRMUX", thus completing the operation.

<u>Host Program</u>	<u>Host</u>		Remote
Write Initial - Request	ENQ	>	1010v
RMHBSCC DATA X'1002' RMHID DATA C'X'	TEXT	<>	ACK*
RMHTYP DATA C'R' RMREQ DATA F'6'			
RMICHK DATA C'HOSTRMUX'		<	ACK*
Write End	EOT	>	NOK.
Read Initial - Status	ACK*	<>	ENQ
RMHTYP='S' RMSREQ=6		<	TEXT
RMSFN=-1 RMSRID='REMTRMUX'			
Read Continue - EOT	ACK*	>	
		<	EOT

Figure 19. Communications Flow for the IDCHECK Function

#### PASSTHRU Function

The PASSTHRU function provides the host with an interface which simulates the capabilities of a terminal connected to a Series/1. Through this interface, the host can interact with the Event Driven Executive supervisor by issuing operator commands, or by interacting with a program as if that program was loaded from a terminal on the Series/1. The host's interaction with the supervisor or a program is conducted in a PASSTHRU session.

Most programs which do not require full screen terminal support, including most Event Driven Executive utilities may be used with the PASSTHRU function. Characteristics of programs which prevent programs from running under the PASSTHRU function are discussed in the section "Considerations on Using PASSTHRU" on page 237.

An example of the use of PASSTHRU could be a host program that formats a host terminal to look like a remote Series/1 terminal. The operator on the host system could then interact with the program as if the terminal was on the remote Series/1.

The PASSTHRU function is initiated by the host sending a PASSTHRU request to the utility. After the request is sent, a series of records are exchanged between the host and the utility, similar to the way messages are written to and read from a terminal. This procedure will be discussed in two parts:

- Establishing a PASSTHRU Session
- Conducting a PASSTHRU session

# Establishing a PASSTHRU Session

As was previously discussed, a PASSTHRU function is initiated by the host sending a PASSTHRU request to the utility. The session is established after the host receives a successful Status record and an EOT. The PASSTHRU request may specify (RMPRPGM field) one of two ways of establishing a session:

- Communication with the Event Driven Executive supervisor
- Communication with a program which the utility will load

If a session with the supervisor is established, the utility will issue an "attention" (as if the attention key on the terminal was pressed). Following the attention, the PASSTHRU session will be conducted with the terminal on the host receiving the caret symbol (>), and continued by the operator entering an operator command, for example \$L.

If a session with a program is established, the host specifies the name of the program and the program is loaded by the utility. The PASSTHRU session will be conducted with the host interacting with the program.

The following fields must be specified on the PASSTHRU request to establish a PASSTHRU session:

- RMHBSCC A 2-byte hexadecimal field containing the BSC control characters DLE STX, specified as X'1002'.
- RMHID A 1-byte alphameric field containing the header ID 'X', identifying the record as an Event Driven Executive Remote Management Utility record.
- RMHTYP A 1-byte alphameric field identifying the header type. This field contains the character 'R', specifying a Request record type.
- RMREQ A 2-byte numeric field specifying the request type. For a PASSTHRU request, this field contains the number 12.
- RMPRBLK A 2-byte numeric field indicating whether the host is to receive blocked records from the remote. A value of O specifies that records are unblocked. A value greater than O specifies the size, in bytes, of the record block (size of "Text or PF Key" extension after the RMPTYP field). See the section "PASSTHRU Blocking" on page 237 for details on this field.
- RMPRFLG A 1-byte reserved field (unused).
- RMPRPTN A 1-byte numeric field specifying the partition the program is to run in. Specify one of the following:
  - -1 Remote Management Utility partition
  - 0 Any partition
  - 1-8 Specific partition
- RMPRPGM An 8-byte alphameric field specifying the name of the program or utility to interact with the host. If this field is blank, a session with the Event Driven Executive is established.
- RMPRVOL A 6-byte alphameric field specifying the name of the volume which contains the program or utility. If blank, the name defaults to the IPL volume name.

- RMPRLFS A 2-byte numeric field specifying the amount of free space (in bytes) to pass to the program.
- RMPRPRM# A 2-byte numeric field specifying the length of the parameter(s), in words, to pass to the program. This field must be zero if no parameters are passed.
- RMPRPRM A variable length field containing the parameter(s) to be passed to the program. The length of this field, in words, must correspond to the value contained in the RMPRPRM# field. See the section "Parameter Passing" on page 212 for details on this field.
- RMPRDS# A 2-byte numeric field specifying the length of data sets to pass to the program. The maximum number of data sets that may be specified is nine. This field must be zero if no data sets are passed.
- RMPRDS A variable number of 14-byte alphameric fields specifying the data set and volume names to be passed to the program. The first eight bytes contain the data set name, and the last six bytes contain the volume name. If the volume name is blank, the name of the volume defaults to the IPL volume. The number of data set and volume names specified must correspond to the value contained in the RMPRDS# field.

Figure 22 on page 241 illustrates the host-remote interaction in establishing a PASSTHRU session.

## Conducting a PASSTHRU Session

Once the PASSTHRU session is established, the session is conducted with Passthru type records exchanged between the host and the remote Series/1. The Passthru records provide information to and receive information from the host program, as if the host program were a terminal on the remote Series/1. Four Passthru records are defined to provide this information. These records are described as follows:

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

The content and format of these records is discussed in the section "Passthru Record Types" on page 232.

Figure 20 on page 230 graphically illustrates how a PASSTHRU session is conducted. In this illustration, each vertical line represents a "state" the host may be in at any time during the session. The name attached at the top of each vertical line is the name of the state. The state of the host may change by one of the following:

- Receiving a Passthru record from the utility. This is represented by a solid horizontal line with an arrow pointing to the new state.
- Sending a Passthru record to the utility. This is represented by 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" as shown in the figure. The host issues a "read" to the communications line and will receive either a "Text or PF Key", "Request for Data", or "Program End" record. The type of record the host receives is determined by the terminal activity occurring in the remote Series/1.

If the host receives a Text or PF Key record, data is being sent to the host. The program (or the supervisor) has issued a PRINTEXT or other terminal I/O instruction, and the message is transmitted to the host as if the host were a terminal. As shown in the figure, the state of the host changes from "READTEXT" to "READING" because the host received a Text or PF Key record. The state then changes back to "READTEXT". Effectively the host remains in the "READTEXT" state as long as Text or PF Key records are received.

If the host receives a Request for Data record, data is needed 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. As shown in the figure, 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 read the EOT also.

In the host's current state, "PGM NEEDS DATA", the host must send a Text or PF Key record followed by an EOT. The Text or PF Key record the host sends may contain either text or a PF key (the host, as a terminal, has entered text or a program function key in response to Request for Data).

If the host sends text, the state of the host changes from "PGM NEEDS DATA" back 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. This Request for Data is sent 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 is then back to the state "READTEXT".

The last possibility from the state "READTEXT" is that the host will receive a Program End record, followed by an EOT. This indicates either 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 was with a program and the program has ended (while in the "CONTINUE ?" state), the host would most likely decide not to continue. If the session was with the supervisor and a \$L command was successfully entered, the host would most likely decide to continue the session and communicate with the program which 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 is now terminated and the Remote Management Utility will wait for a new request from the host. To continue the session, the host should send a Request for Data record followed by an EOT. The state of the host then changes from "CONTINUE?" to "ACTIVITY?".

At this point, the utility determines if there is any terminal activity on the remote Series/1 for the host. If there is activity, one of the three Passthru records which can be received from the "READTEXT" state will be received by the host. These three records are Text or PF Key, Request for Data, or Program End. The state of the host will change as it would from the state "READTEXT".

If there is no terminal activity, the host will receive a No Data record followed by an EOT, and the host's state changes from "ACTIVITY?" to "CONTINUE?". The host may then determine again whether it should continue. If the program in the remote Series/1 has any 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 ever occurs, the host must eventually send a Program End record and terminate the PASSTHRU session.

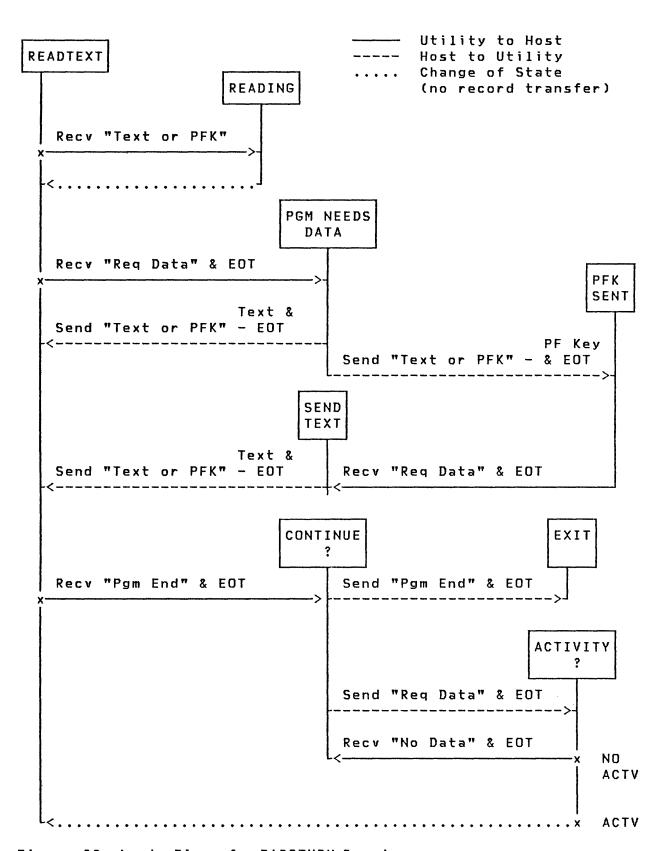


Figure 20. Logic Flow of a PASSTHRU Session

The preceding discussion and Figure 20 on page 230 summarizes the flow of a PASSTHRU session. The only addition to this is that of a severe error being encountered, in which case the host may receive or send a Status record followed by an EOT. An example of where this error condition could occur is if the host sends an invalid Passthru record. The utility will respond to this invalid record with a Status record. Similarly, the host may send a 4-byte Status record (preceded by "abort" if necessary). In either case, the PASSTHRU session is terminated and the utility will wait for a new request.

The following is the format of the Status record sent by the host:

RMHBSCC DATA X'1002'
RMHID DATA C'X'
RMHTYP DATA C'S'

## Passthru Record Types

This section describes in detail the format and content of the four types of Passthru records previously mentioned.

# Text or Program Function Key

This record is comprised of two segments. The first six bytes, or the main segment, identifies this record as a Passthru Text or Program Function (PF) Key record. Following the main segment is one or more text or PF key segments. The following is an illustration of these two segments:

## 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'

In the main segment, all values are constants as shown. The text or program function key segment contains the information to be transferred:

RMPST A 2-byte value of the return code. This field contains a value only on records received by the host.

RMPTXTL A 2-byte numeric field specifying either the length of the text, or indicating a PF key is being sent.

RMPTXT Either a variable-length alphameric field containing text, or a 2-byte numeric field containing the PF key value.

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 more than one text or program function key segments. All records sent by the host are unblocked. Records received by the host may be blocked if specified on the PASSTHRU request. Details on how to specify blocking is discussed in the section "PASSTHRU Blocking" on page 237.

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 program function 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 illustrates a record sent by the host which contains the text "MESSAGE FROM HOST PROGRAM":

Text record sent by the host:

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

When the host sends a program function key, the value of the RMPTXTL field is set to -1 and the program function 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 illustrates a program function key 3 being sent by the host:

Program function key record sent by host:

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

All Text or PF Key records received by the host will always contain text; the host will never receive a program function key. Each Text or PF Key record begins with the 6-byte main segment followed by one or more text segments. The fields in each text segment are defined as follows:

RMPST A 2-byte numeric containing the return code associated with the text. For example, the return code indicates whether the text is to appear on a new line. Some return codes have no text associated with them. For a complete description of the possible return codes, refer to virtual terminal communications return codes as described for the READTEXT instruction in the Language Reference.

The return codes which are applicable 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 containing the text length. If there is no text, this field will contain the value 0.

RMPTXT A variable-length alphameric field containing the text received by the host. The length of this field, in bytes, is the value of RMPTXTL. If RMPTXTL is an odd number, one byte of blanks (X'40') follows the text.

If records are blocked, multiple text segments are received on a Text or PF Key record. The host must determine the length of the record in order to process each segment. Figure 21 on page 235 is an example of the records the host receives from a program which executes a PRINTEXT instruction.

```
Issued by program on remote Series/1:
  PRINTEXT 'ENTER COMMAND', SKIP=1
Passthru record received by host
with no blocking:
  RMHBSCC
           DATA
                 X'1002'
                 C'X'
  RMHID
           DATA
                 C'P'
  RMHTYP
           DATA
  RMPTYP
           DATA
                 F'1'
           DATA
                 X'8E01'
  RMPST
                           (SKIP=1)
  RMPTXTL
           DATA
                F'0'
                           (NO TEXT)
  RMHBSCC
           DATA
                X'1002'
           DATA
                C'X'
  RMHID
  RMHTYP
           DATA
                C'P'
                F'1'
  RMPTYP
           DATA
           DATA
                F'-2'
  RMPST
  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'
                 C'P'
  RMHTYP
           DATA
  RMPTYP
           DATA
                 F'1'
           DATA
                 X'8E01'
                           (SKIP=1)
           DATA
                F'0'
                           (NO TEXT)
                F'-2'
                           (NEXT SEGMENT)
           DATA
           DATA
                F'13'
                 C'ENTER COMMAND'
           DATA
           DATA
                 C''
                           (PAD)
```

Figure 21. Example of Passthru Records Received by Host

# Request for Data

The Request for Data record is a 6-byte record which contains constant values. A Request for Data record is always followed by an EOT.

The following is the format of the Request for Data record:

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

# Program End

The Program End record is a 6-byte record which contains constant values. A Program End record is always followed by an EOT.

The following is the format of the Program End record:

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

# No Data

The No Data record is a 6-byte record which contains constant values. A No Data record is always followed by an EOT.

The following is the format of the No Data record:

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

#### PASSTHRU Blocking

When Passthru records are blocked, the communications line is used more efficiently. Without blocking, each Text or PF Key record contains only one text segment. With blocking, each record may contain multiple text segments. Through use of blocking, the amount of information and the number of records transmitted over the communications line is reduced. Thus blocking allows more efficient usage of the communications line, especially for PASSTHRU sessions in which the host receives many consecutive lines of output, such as a result of a "list" command to a utility.

To use PASSTHRU blocking, 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.

The host specifies blocking on the PASSTHRU request in the RMPRBLK field. If this field is set to zero blocking is not performed. A value greater than zero indicates the maximum length 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 blocking size the host may use. This number would then be specified in the RMPRBLK field of the PASSTHRU request. The utility will use this value if it can. If, however, the utility does not have a buffer of sufficient size to provide records of the size requested, the utility will block to the largest size it can handle. Refer to the section "Modifying Defaults" on page 283 for additional information on the maximum blocking size of the utility.

If a single text record should exceed the size specified for RMPRBLK, the utility will send 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 length is 132 bytes, a minimum block size of 136 would be sufficient for all records.

### Considerations on Using PASSTHRU

As mentioned earlier, most programs can be used with the PASSTHRU function of the Remote Management Utility. In this section, considerations on the use of the PASSTHRU function are discussed. These include a discussion of restrictions on the use of the PASSTHRU function and programming techniques.

The PASSTHRU function uses the virtual terminal support of the Event Driven Executive, and therefore has any restrictions inherent in this support. The primary one is that static screens are not supported, therefore programs requiring static screens can not be run under the PASSTHRU function. This includes such programs as the full screen editor, \$FSEDIT. Another restriction is that message length may be no longer than 254 bytes.

The utility allows the host to transmit a program function key or an attention only when the remote is already requesting data. Therefore output from the remote may not be "interrupted" by an "attention", as it could be on a local terminal. For example, a listing produced by the \$DISKUT2 utility could not be interrupted by an "attention" and cancel command.

If a program stops communicating with the terminal which loaded it, and waits on the terminal to enter commands by way of "attention" or program function keys, it will not run directly under the PASSTHRU function. This occurs because the Remote Management Utility will wait indefinitely on a "READTEXT" to the virtual channel while the remote program is waiting on an attention or PF key. When this happens, this is referred to as a deadlock situation. Programs which do this include the following:

```
$DEBUG

$TRAP

$LOG

$BSCTRCE

$TERMUT3 (Attention-entered commands)

$IOTEST (Attention-entered commands)

CALCDEMO (Sample program)
```

A program has been provided which will break the deadlock situation when it occurs. The program name is \$RMUPA. It must be started under the PASSTHRU function prior to starting a PASSTHRU session with one of the programs which may have this problem. \$RMUPA will cause a "disconnect", resulting in a Program End Passthru record being received at the host whenever the following sequence of events occurs:

- No activity has occurred over the virtual channel for 20 seconds.
- The utility is waiting on completion of a "READTEXT" instruction.
- 3. The remote program is not ENQT'ed on its virtual terminal.

The program uses the STIMER instruction, and therefore requires timer support to be included in the remote system.

Due to a timing situation when multiple programs are communicating over a virtual channel, blocking must be used while running these programs.

The sample PASSTHRU host program in the section "Sample Host Programs" on page 259 illustrates how to use the program \$RMUPA from a host program. \$RMUPA is first started under the PASSTHRU function. 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 is terminated. Only one copy of \$RMUPA should be running at a time. It may run in any partition. It continues running until an "attention" follower by "\$RMUPA" is entered.

Once \$RMUPA is running, another program may be started. The sample PASSTHRU host program interaction in the section "Sample Host Programs" on page 259 illustrates how \$DEBUG may be used. Note that "\$PFO" is entered to provide the same function as entering the "attention" key.

If a remote program should take longer than 20 seconds between performing terminal I/O, RMUPA will cause 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.

If a program is run under the PASSTHRU function which issues an ENQT instruction for a terminal other than the terminal which loaded the program and the program terminates, the utility does not receive a "disconnect" over the virtual channel and the host will not receive a Program End record. The utility will wait indefinitely. One example of where this will occur is in running \$EDXASM, 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 on the program to complete.
- Load the program through a session with the Event Driven Executive via a \$L command and respond with a Program End when the command terminates. Programs requiring terminal interaction after being loaded, such as \$EDXASM, will not work in this manner, so should be handled in the first way.

When multiple programs are communicating over a virtual channel, blocking must be used. As mentioned previously, this is due to a timing situation with multiple programs.

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

In the event a PASSTHRU session is abruptly terminated (status received from host, invalid message received from host, or an error in the BSC), the utility will cause a terminal I/O return code 5 ("Disconnected") to be received by the program for the outstanding terminal request. This code will only be received once by the PASSTHRU-invoked program, and the program should then take appropriate action, which would most likely be to terminate. However, if the program does not recognize the terminal error and continues to perform terminal I/O, the program will interfere with attempts to establish a new PASSTHRU session. If the new session is being established with a program, the utility will return the status "virtual terminal busy". The host may establish a session with the Event Driven Executive and issue a \$C command to cancel the suspended program. The \$C command should be used with caution, as noted in the Utilities, Operator Commands, Program Preparation, Messages and Codes.

When a \$L command is issued during a PASSTHRU session with the Event Driven Executive supervisor, a Program End Passthru 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.

As described in the <u>System Guide</u>, two virtual terminals, named CDRVTA and CDRVTB, must be defined for using the PASSTHRU function. Also, virtual terminal support must be included at system generation time. Refer to the <u>System Guide</u> for details.

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

- Avoid any long delays at the host while messages are being received from the remote Series/1.
- Define a high retry count for the RETRIES parameter of the BSCLINE statement in the remote system.

Figure 22 on page 241 illustrates the host-remote interaction for the PASSTHRU function. In the example, the host specifies the program "MYPROG" on the volume "MYVOL" is to be executed. While executing, the program writes one message to the virtual terminal via a Passthru record, receives one message from the virtual terminal via a Passthru record, and terminates.

Host Program			<u>Host</u>		Remote
Write Ini	itial	- Request	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'O'			
RMPRPRM	EQU	<b>*</b>			
RMPRDS#	DATA	F'0'			
RMPRDS	EQU	×			
				<	ACK*
Write End	t		EOT	>	
Read Init	tial -	Status		<	ENQ
			ACK×	>	
RMHTYP='S	3 <b>'</b>			<	TEXT
RMSREQ=12	2				
RMSFN=-1					
Read Cont	tinue	- EOT	ACK*	>	
				<	EOT
				,	

Figure 22. Communications Flow for the PASSTHRU Function (Part 1 of 2)

Read Initial - Passthru Data		<	FNO
, was an in the same	V C K *	>	-
RMHTYP='P'	ACK A	<	
RMPTYP=1		(	IEXI
RMPST=Status from READTEXT			
RMPTXTL=Message length			
RMPTXT=Message text			
Read Continue - Request for Data	ACK×	>	
RMHTYP='P'		<	TEXT
RMPTYP=2			
Read Continue - EOT	ACK*	>	
		<	FOT
   Write Initial - Passthru Data	ENO	>	LU1
HILLE THICIAL . LASSCHIN DACK	E IY V	<	ACVX
RMHTYP='P'	TEVT	>	ACK*
	IEXI	>	
RMPTYP=1			
RMPST=0 (Unused)			
RMPTXTL=Message length			
RMPTXT=Message text			
		<	ACK*
Write End	EOT	>	
Read Initial - Passthru Program E	End	<	ENQ
		>	
RMHTYP='P'		<	
RMPTYP=3		•	ILXI
NIII   11 = 3			
Pand Continue - FOT	ACVV	>	
Read Continue - EOT	ACK	<	
		·	EUI
Write Initial - Passthru Program			
	ENQ	>	
		<	ACK*
RMHTYP='P'	TEXT	>	
RMPTYP=3			
		<	ACK*
Write End	EOT	>	

Figure 23. Communications Flow for the PASSTHRU Function (Part 2 of 2)

#### RECEIVE Function

The RECEIVE function requests the utility to receive a data set transmitted from the host and to write it to a disk/diskette data set on the remote Series/1.

The host can specify it is sending a data set consisting of 256-byte data records, or a source data set, consisting of 80-byte text records. The host may also specify blocking, in which case, the utility receives records containing multiples of 256-byte or 80-byte records.

The host sends the remote Series/1 a Request record with the RECEIVE function specified. After receiving and executing the RECEIVE request, the utility checks to see if it can handle records of the size requested and attempts to open the data set. The utility then sends a Status record to the host. If a -1 (successful) status is returned to the host the RECEIVE function continues, otherwise the function is terminated.

Upon receipt of the successful status, the host sends records to the utility. The data contained within the records sent by the host should have a length which is a multiple of 256 or 80, depending on the data set type. If the utility receives a record whose length is not a multiple of 256 or 80 (short record), the record is padded with zeroes, and then written to disk or diskette. For example, assume that a 256-byte record data set with a blocking factor of 3 is specified. A record received with with a length of 256 will cause one record to be written. A record received with a length of 512 will cause two records to be written, and similarly, a length of 768 will cause three records to be written, all with no padding. However, a record received with a length of 300 would cause two records to be written. The first containing the first 256 bytes of data, and the second containing the last 44 bytes of data followed by 212 zeroes (X'00'), thus padding it to a length of 256 bytes.

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

At the completion of the data set transfer, the utility performs a SETEOD on the data set, and sends the host a Count record. The Count record specifies the number of records received in the RMCCNT field, and if padding occurred at any time, the RMCFLGPD bit of the RMCFLG field is set to 1. The RMCFLGPD bit is defined by the value X'8000'.

If the data set to be received by the utility is empty, the host should send one Data record which contains no data (only the 4-byte header), and then the EOT.

In the event of unrecoverable errors, such as disk or diskette errors, the utility interrupts the host transmission by sending an EOT ("abort") and a Status record containing the appropriate error code. The utility terminates the RECEIVE operation, and then reads again for another request from the host. The host should accept the Status record to determine the reason for failure.

The host may terminate the RECEIVE function at any time by sending a Status record followed by an EOT.

The RECEIVE function has no restrictions on receiving data sets with names such as \$EDXNUC, \$\$EDXVOL, or \$\$EDXLIB. However, care should be exercised if these data sets are transferred. As was previously mentioned, a SETEOD is performed upon completion of a data set transfer. The SETEOD may not be performed on data sets with the names \$\$EDXVOL or \$\$EDXLIB. Thus, if these data sets are transferred, the host will receive a Status record indicating a SETEOD error. Additionally, SETEOD will fail if the data set type is "program". This failure is ignored by the Remote Management Utility.

### Required Field Descriptions

Specify the following fields for the RECEIVE function:

- RMHBSCC A 2-byte hexadecimal field containing the BSC control characters DLE STX, specified as X'1002'.
- RMHID A 1-byte alphameric field containing the header ID 'X', identifying the record as an Event Driven Executive Remote Management Utility record.
- RMHTYP A 1-byte alphameric field identifying the header type. This field contains the character 'R', specifying a Request record type.
- RMREQ A 2-byte numeric field specifying the request type. For a RECEIVE request, this field contains the number 1.
- RMRDSN An 8-byte alphameric field specifying the name of the data set to receive data from the host.
- RMRVOL A 6-byte alphameric field specifying the name of the volume containing the data set. If RMRVOL is blank, the volume name defaults to the IPL volume.

RMRSTR A 4-byte (double word) numeric field specifying the starting record of the host data set. Only the second word of this field is used. If a value of 0 is specified, the data set is received and written from the beginning record. If a value greater than zero is specified, the utility issues a POINT instruction and starts receiving data at the record specified.

RMRTYP A 2-byte numeric field specifying the type of data to be received. Specify one of the following:

- O Standard (256-byte records, possibly blocked)
- 1 Source (80-byte records, possibly blocked)

RMRBLK A 2-byte numeric field specifying blocking. A value of 0 or 1 specifies no blocking; otherwise it specifies the number of 80-byte or 256-byte records to be received on each Data record.

Figure 24 on page 246 illustrates the host-remote interaction for the RECEIVE function. In the example, the host specifies a data set named "MYDATA" on the volume "MYVOL" is to receive two 256-byte data records. The records to be received start at the beginning of the host data set, and are unblocked. The remote returns a Count record, and the RECEIVE function terminates.

245

Write Initial - Request				<i></i>
RMHBSCC DATA	Host Program	<u>Host</u>		Remote
RMHBSCC DATA X'1002' RMHID DATA C'X' RMHID DATA C'R' RMREQ DATA F'1' RMRDSN DATA CL6'MYVOL' RMRSTR DATA D'0' RMRTYP DATA F'0' RMRTYP DATA F'1'  Write End EOT>  Read Initial - Status	Write Initial - Request	ENQ		V C K *
RMHTYP DATA C'R' RMREQ DATA F'1' RMRDSN DATA CL8'MYDATA' RMRVOL DATA CL6'MYVOL' RMRSTR DATA D'O' RMRTYP DATA F'O' RMRBLK DATA F'1'  Write End EDT		TEXT		AVK.
RMREQ DATA F'1' RMRDSN DATA CL8'MYDDATA' RMRYOL DATA CL6'MYVOL' RMRSTR DATA D'0' RMRTYP DATA F'0' RMRBLK DATA F'1'  Write End EDT				
RMRDSN DATA CL8'MYDATA' RMRVOL DATA CL6'MYVOL' RMRSTR DATA D'O' RMRTYP DATA F'O' RMRBLK DATA F'O' RMRBLK DATA F'I'  Write End EOT	RMHTYP DATA C'R'			
RMRVOL DATA CL6'MYVOL' RMRSTR DATA D'0' RMRTYP DATA F'0' RMRBLK DATA F'1'  Write End EDT	RMREQ DATA F'1'			
RMRSTR DATA D'0' RMRTYP DATA F'0' RMRBLK DATA F'1'  Write End EDT				
RMRTYP       DATA       F'0'         RMRBLK       DATA       F'1'         Write       EOT				
RMRBLK       DATA       F'1'       <				
Write End	RMRTYP DATA F'O'			
Write End       EOT      >         Read Initial - Status       < ENQ	RMRBLK DATA F'1'			
Read Initial - Status       < ENQ			<	ACK×
RMHTYP='S'       ACK*	Write End	EOT	>	
RMHTYP='S'       ACK*				
RMHTYP='S' RMSREQ=1 RMSFN=-1  Read Continue - EOT  Write Initial - Data  ENQ  EOT  White Initial - Data  ENQ  RMHBSCC DATA X'1002' RMHID DATA C'X' RMHTYP DATA C'D' RMDDATA DATA C text  Write Continue - Data RMHBSCC DATA X'1002' RMHID DATA C'D' RMHIP DATA C'D' RMDDATA DATA C text  Write End  EOT  ACK*  Write End  Read Initial - Count  ACK*  TEXT	Read Initial - Status		<	ENQ
RMHTYP='S'       <		ACK*		•
RMSREQ=1 RMSFN=-1  Read Continue - EOT	RMHTYP='S'		<	TEXT
RMSFN=-1  Read Continue - EOT				
Read Continue - EOT       ACK* EOT         Write Initial - Data       ENQ ACK*         RMHBSCC DATA X'1002'       TEXT ACK*         RMHID DATA C'X'       RMHTYP DATA C'D'         RMBSCC DATA X'1002'       TEXT ACK*         Write Continue - Data RMHBSCC DATA X'1002'       TEXT ACK*         RMHID DATA C'X'       RMHTYP DATA C'D'         RMHTYP DATA C'D'       C'D'         RMDDATA DATA C text       C ACK*         Write End       EOT ACK*         Read Initial - Count       C ENQ         ACK* TEXT         RMHTYP='C'       C TEXT				
C EOT				
C EOT	Read Continue - FOT	ACK*	>	
Write Initial - Data       ENQ ACK*         RMHBSCC DATA X'1002'       TEXT>         RMHID DATA C'X'       TEXT>         RMHTYP DATA C'D'       C text         Write Continue - Data       TEXT>         RMHBSCC DATA X'1002'       TEXT>         RMHID DATA C'X'       TEXT>         RMHTYP DATA C'D'       C text         Write End       EOT         Read Initial - Count       ACK*	nead out that Est			FOT
C	Write Initial - Data	FNO		201
RMHBSCC       DATA       X'1002'       TEXT      >         RMHID       DATA       C'X'       C'D'       C'D'       C'TO'       C'	Alle Inicial Data	L. 11 Q		V C K *
RMHID       DATA       C'X'         RMHTYP       DATA       C'D'         RMDDATA       DATA       C'ext         Write       Continue       - Data         RMHBSCC       DATA       X'1002'         RMHID       DATA       C'X'         RMHTYP       DATA       C'D'         RMDDATA       DATA       C text         Write       EOT	DMUDECC DATA VIIONOI	TEYT		MOK^
RMHTYP       DATA       C'D'         RMDDATA       DATA       C text         Write       Continue       - Data         RMHBSCC       DATA       X'1002'         RMHID       DATA       C'X'         RMHTYP       DATA       C'D'         RMDDATA       DATA       C text         Write       EOT		ILAI		
RMDDATA       DATA       C text       < ACK*				
C				
Write Continue - Data RMHBSCC DATA X'1002' RMHID DATA C'X' RMHTYP DATA C'D' RMDDATA DATA C text  Write End  Read Initial - Count  RMHTYP='C' RMCREQ=1  TEXT>  TEXT>  ACK*	KUDDATA DATA C CEXC		/	ACVX
RMHBSCC DATA X'1002'       TEXT>         RMHID DATA C'X'       RMHTYP DATA C'D'         RMDDATA DATA C text       < ACK*	Unite Centinus - Data		<b>(</b>	ACK*
RMHID       DATA       C'X'         RMHTYP       DATA       C'D'         RMDDATA       DATA       C text         Write       EDT		TEVT		
RMHTYP       DATA       C'D'         RMDDATA       C text       < ACK*		IEXI		
RMDDATA DATA C text       < ACK*				
Count   Coun				
Write End	RMDDAIA DAIA C text		_	104
Read Initial - Count			<	AUK*
Read Initial - Count		FAT	_	
RMHTYP='C'	Write End	EUT	>	
RMHTYP='C'			_	
RMHTYP='C' < TEXT RMCREQ=1	Read Initial - Count			FNQ
RMCREQ=1		ACK*		
			<	TEXT
DMOONT - 3				
KMCCN1=2	RMCCNT=2			
Read Continue - EOT ACK*>	Read Continue - EOT	ACK*		
< EOT			<	EOT

Figure 24. Communications Flow for the RECEIVE Function

#### SEND Function

The SEND function requests the utility to read a disk/diskette data set on the remote Series/1 and transmit it to the host.

The host can specify whether it wants a data set consisting of 256-byte data records, or a source data set, consisting of 80-byte text records sent from the remote. The host may also specify blocking, in which case, the utility sends records containing multiples of 256-byte or 80-byte records.

The host sends the remote Series/1 a Request record with the SEND function specified. After receiving and executing the request, the utility checks to see if it can handle records of the size requested and attempts to open the data set. The utility then sends a Status record to the host. If a -1 (successful) status is returned to the host the SEND function continues, otherwise the function is terminated.

After sending a successful status to the host, the remote Series/1 reads the records from the data set and transmits Data records containing the data to the host. If blocking is specified, the utility sends blocked Data records to the host. The length of the data portion of each Data record, except for the last, will be the blocking factor times 256 or 80, depending on the data set type. The data portion of the last Data record will have a length of a multiple of 256 or 80, however that multiple may be less than the blocking factor. For example, if a 256-byte record data set contains 14 records and a blocking factor of 5 is specified, the utility will send two 1285-byte (256x5=1280+5), and one 1029-byte (256x4=1024+5). The actual records are five bytes longer due to the 4-byte header and the ETX.

If the host requests a data set to be sent as source (80-byte records) and the data set is not source, the utility will treat the data set as source, and discard the remaining 48-bytes following the 80-byte records.

When the last record (the logical end) of the data set is transmitted to the host, the utility will send a Count record. The RMCCNT field of the Count record contains the number of records that were sent. The RMCFLG field of the Count record is not used for the SEND function. The host should compare this number to the number of records received to verify a complete file transfer.

In the event of an unrecoverable error, such as a disk or diskette read error, the utility sends the host a Status record, with the appropriate error code, and terminates the SEND function. The host may terminate a SEND function by sending an EOT ("abort"), followed by a Status record and another EOT.

### Required Field Descriptions

Specify the following fields for the SEND function:

- RMHBSCC A 2-byte hexadecimal field containing the BSC control characters DLE STX, specified as X'1002'.
- RMHID A 1-byte alphameric field containing the header ID 'X', identifying the record as an Event Driven Executive Remote Management Utility record.
- RMHTYP A 1-byte alphameric field identifying the header type. This field contains the character 'R', specifying a Request record type.
- RMREQ A 2-byte numeric field specifying the request type. For a SEND request, this field contains the number 0.
- RMSDSN An 8-byte alphameric field specifying the the name of the data set to be transmitted to the host.
- RMSVOL A 6-byte alphameric field specifying the name of the volume containing the data set. If RMSVOL is blank, the volume name defaults to the IPL volume.
- RMSSTR A 4-byte (double word) numeric field specifying the starting record of the data set. Only the second word is used. If a value of 0 is specified, the data set is sent beginning with the first record. If a value greater than zero is specified, the utility issues a POINT instruction to start at the record specified.
- RMSTYP A 2-byte numeric field specifying the type of data set to send. Specify one of the following:
  - O Standard (256-byte records, possibly blocked)
  - 1 Source (80-byte records, possibly blocked)
- RMSBLK A 2-byte numeric field specifying blocking. A value of 0 or 1 specifies no blocking; otherwise it specifies the number of 80-byte or 256-byte records to be transmitted on each Data record.

Figure 25 on page 250 illustrates the host-remote interaction for the SEND function. In the example, the host requests that a 256-byte record data set named "MYDATA" on the volume "MYVOL" is to be sent, starting with the first record, with no blocking requested. The utility transmits three Data records, sends a Count record to the host, and the SEND function terminates.

<u>Host Program</u>	<u>Host</u>		Remote
Write Initial - Request	ENQ	>	
		<	ACK*
RMHBSCC DATA X'1002'	TEXT	>	
RMHID DATA C'X'			
RMHTYP DATA C'R'			
RMREQ DATA F'O'			
RMSDSN DATA CL8'MYDATA'			
RMSVOL DATA CL6'MYVOL'			
RMSSTR DATA D'O'			
RMSTYP DATA F'O'			:
RMSBLK DATA F'1'		_	
		<	ACK*
Write End	EOT	>	
		_	=110
Read Initial - Status		<	ENQ
	ACK*	>	T-VT
RMHTYP='S'		<	IEXI
RMSREQ=0			
RMSFN=-1			
Read Continue - Data	<b>V C K *</b>	>	
RMHTYP='D'	ACKA	<	TEVT
RMDDATA=Data Text		<b>\</b>	1571
KNDDATA-Data Text			
Read Continue - Data	ACK*	>	
RMHTYP='D'		<	TFXT
RMDDATA=Data Text		•	
Read Continue - Data	ACK*	>	
RMHTYP='D'		<	TEXT
RMDDATA=Data Text			
Read Continue - Count	ACK*	>	
RMHTYP='C'		<	TEXT
RMCREQ=0			
RMCCNT=3			
Read Continue - EOT	ACK×		
		<	EOT
	·····		

Figure 25. Communications Flow for the SEND Function

#### SHUTDOWN Function

The SHUTDOWN function requests the utility to terminate and to free up any remote Series/1 resources it has allocated. In addition, the SHUTDOWN function can optionally start a program to replace the utility.

The host sends the remote Series/l a Request record with the SHUTDOWN function specified. The request may also specify the name of a program to be executed, similar in format to the EXEC function.

When a program is specified on the SHUTDOWN request, the utility issues a LOAD instruction for the program. If the LOAD instruction fails, the utility sends the host a Status record indicating the error, and the utility remains active. Otherwise, the utility sends a successful status via a Status record and terminates.

If the program specified by the host requires a parameter and the parameter is not supplied, the load (via LOAD) of the program will fail. The character string is the parameter(s). For further information on parameter passing, refer to the section "Parameter Passing" on page 212.

### Required Field Descriptions

Specify the following fields for the SHUTDOWN function:

- RMHBSCC A 2-byte hexadecimal field containing the BSC control characters DLE STX, specified as X'1002'.
- RMHID A 1-byte alphameric field containing the header ID 'X', identifying the record as an Event Driven Executive Remote Management Utility record.
- RMHTYP A 1-byte alphameric field identifying the header type. This field contains the character 'R', specifying a Request record type.
- RMREQ A 2-byte numeric field specifying the request type. For a SHUTDOWN request, this field contains the number 7.
- filler A 2-byte reserved field (unused).

RMSDFLG A 1-byte numeric field containing the RMSDFLGX and RMSDFLGL bits.

RMSDFLGX - When set on, this bit indicates that a program is to be executed. The value for RMSDFLGX when set on is X'80'.

RMSDFLGL — When set on, this bit indicates that a "program loaded" message is to be printed on the system logging terminal. RMSDFLGL corresponds to the usage of the LOGMSG parameter of the Event Driven Language LOAD instruction. The value for RMSDFLGL when set on is X'40'.

RMSDPTN A 1-byte numeric field specifying the partition the program is to run in. Specify one of the following:

- -1 Remote Management Utility partition
- O Any partition
- 1-8 Specific partition

RMSDPGM An 8-byte alphameric field specifying the name of the program to be executed.

RMSDVOL A 6-byte alphameric field specifying the name of the volume containing the program. If RMSDVOL is blank, the volume name defaults to the IPL volume.

RMSDLFS A 2-byte numeric field specifying the amount of free space (in bytes) to pass to the program.

RMSDPRM# A 2-byte numeric field specifying the length of the parameter(s), in words, to pass to the program. This field must be zero if no parameters are passed.

RMSDPRM A variable length field containing the parameter(s) to be passed to the program. The length of this field, in words, must correspond to the value contained in the RMSDPRM# field. See the section "Parameter Passing" on page 212 for details on this field.

RMSDDS# A 2-byte numeric field specifying the number of data set names to be passed to the program. The maximum number of data set names that may be specified is nine. This field must be zero if no data set names are passed.

RMSDDS A variable number of 14-byte alphameric fields specifying data set and volume names to be passed to the program. The first eight bytes contain the data set name, and the last six bytes contain the volume name. If the volume name is blank, the name of the volume defaults to the IPL volume. The number of data set and volume names specified must correspond to the value contained in the RMSDDS# field.

Figure 26 illustrates the host-remote interaction for the SHUTDOWN function. In the example, the host sends the remote a SHUTDOWN request with a program name specified. The program, "MYPROG" on the 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. The remote sends a status of -1 (successful) to the host, loads the program, and the utility terminates itself.

<u>Host Program</u>			<u>Host</u>	Remote	
Write Ini	itial	- Request	ENQ	>	
DMUDGGG	DATA	VI10031	TEVT	<	ACK*
RMHBSCC RMHID	DATA	X'1002'	IEXI	>	
	DATA				
RMREQ		•			
RMSDFLG RMSDPTN					
		CL8'MYPROG'			
		CL6'MYVOL'			
RMSDFLS					
RMSDPRM#					
RMSDPRM					
RMSDDS#					
RMSDDS#					
KIISDDS	LQU	^		<	V C K *
Write End	<del>j</del>		EOT	>	AUKA
Read Init	tial -	Status		<	ENQ
			ACK*		
RMHTYP='S'				<	TEXT
RMSREQ=7					
RMSFN=-1					
Read Cont	tinue	- FOT	ACK*	>	
nead oom	. , mac		DVKA	<	FOT

Figure 26. Communications Flow for the SHUTDOWN Function

253

## WRAP Function

The WRAP function requests the utility to send a block of data just received back to the host.

The host sends the remote Series/1 a Request record with the WRAP function specified. The text to be wrapped (transmitted) is specified in the RMWTXT field of the record extension. The utility transmits the Request record including the text back to the host exactly as it was received, and the function terminates. The utility does not send a Status record to the host after execution of a WRAP function.

A possible use of the WRAP function could be for testing the host/remote communications.

### Required Field Descriptions

Specify the following fields for the WRAP function:

- RMHBSCC A 2-byte hexadecimal field containing the BSC control characters DLE STX, specified as X'1002'.
- RMHID A 1-byte alphameric field containing the header ID 'X', identifying the record as an Event Driven Executive Remote Management Utility record.
- RMHTYP A 1-byte alphameric field identifying the header type. This field contains the character 'R', specifying a Request record type.
- RMREQ A 2-byte numeric field specifying the request type. For a WRAP request, this field contains the number 5.
- RMWTXT A field of any length (not greater than the buffer) specifying text to be transmitted back to the host.

Figure 27 illustrates the host-remote interaction for the WRAP function. In the example, the host sends the remote a WRAP request along with the text "WRAP TEXT" specified. The remote receives the request and transmits the identical request back to the host, and the operation is completed.

	···	<del></del>	
Host Program	<u>Host</u>		Remote
Write Initial - Request	ENQ	>	1 C K *
RMHBSCC DATA X'1002' RMHID DATA C'X'	TEXT	>	AUKA
RMHTYP DATA C'R' RMREQ DATA F'5'			
RMWTXT DATA C'WRAP TEXT'		<	ACK*
Write End	EOT	>	
Read Initial - Wrap	ACK*	<>	ENQ
RMHBSCC=X'1002' RMHID=C'X' RMHTYP=C'R'	no.	<	TEXT
RMREQ=F'5' RMWTXT=C'WRAP TEXT'			
Read Continue - EOT	ACK*	> <	EOT

Figure 27. Communications Flow for the WRAP Function

### Count Record

The Remote Management Utility sends a Count record to the host after an end-of-data condition is detected during a data set transfer (from either a SEND or RECEIVE request). This record contains the number of records sent or received by the utility. Additionally, the Count record indicates if record padding has occurred during the data set transfer. The host should use this record to verify whether a complete file transfer has occurred.

The following is the format of the Count record:

- RMHBSCC A 2-byte hexadecimal field containing the BSC control characters DLE STX, specified as X'1002'.
- RMHID A 1-byte alphameric field containing the header ID 'X', identifying the record as an Event Driven Executive Remote Management Utility record.
- RMHTYP A 1-byte alphameric field identifying the header type. This field contains the character 'C', specifying a Count record type.
- RMCREQ A 2-byte numeric field specifying the request type (0=SEND, 1=RECEIVE).
- RMCFLG A 2-byte field indicating if record padding has occurred during a data set transfer. The bit defined by RMCFLGPD (X'8000') is set to 1 if padding has occurred, otherwise 0.
- RMCCNT A 4-byte numeric field specifying the number of records transmitted. This number reflects the number of logical records (80-byte or 256-byte records) transmitted, independent of how the records were blocked.

#### Data Record

The Data record is used by the Remote Management Utility to send data to or receive data from the host. This record contains the 80-byte or 256-byte records from a specified data set on a SEND or RECEIVE request.

The following is the format of the Data record:

- RMHBSCC A 2-byte hexadecimal field containing the BSC control characters DLE STX, specified as X'1002'.
- RMHID A 1-byte alphameric field containing the header ID 'X', identifying the record as an Event Driven Executive Remote Management Utility record.
- RMHTYP A 1-byte alphameric field identifying the header type. This field contains the character 'D', specifying a Data record type.
- RMDDATA A variable-length field containing the data to be transmitted (from a SEND or RECEIVE request). The length of this field will be a multiple of 80 or 256, depending on the type of data transfer.

#### Status Record

The Status is sent to the host by the Remote Management Utility to indicate the success or failure of a requested function.

The following is the format of the Status record:

- RMHBSCC A 2-byte hexadecimal field containing the BSC control characters DLE STX, specified as X'1002'.
- RMHID A 1-byte alphameric field containing the header ID 'X', identifying the record as an Event Driven Executive Remote Management Utility record.
- RMHTYP A 1-byte alphameric field identifying the header type. This field contains the character 'S', specifying a Status record type.
- RMSREQ A 2-byte numeric field specifying the request type.
- RMSFN A 2-byte numeric field indicating the success of the request. If the request is successful this field will contain a -1, otherwise this field will contain a positive value indicating the error which occurred. The equated values, included in the copy code CDRRM, with the names beginning with the RMSFN field define these errors.
- RMSST A 2-byte numeric field with a return code if an Event Driven Executive function failed. For example, if RMSFN contained the value 24 (LOAD failed), RMSST will contain the return code from the LOAD instruction.
- RMSRID An 8-byte alphameric field specifying the ID of the remote Series/1 on completion of a successful IDCHECK request. This field is not sent to the host if the IDCHECK request fails.

# Sample Host Programs

The following sample programs illustrate host programs (on a host Series/1) which can communicate with and perform functions of the Remote Management Utility.

This sample host program can perform all the functions of the utility except SEND, RECEIVE, and PASSTHRU. This program sends an ALLOCATE request and prints a status message, but could be used for the other functions by simply defining the fields of the desired request at label "RM".

UT Start	PROGRAM S	START *	
• • • • • • • • • • • • • • • • • • • •		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		IMEOUT=NO READ STATUS
	SUB	IOCB, IOCB2, RESULT=PN2	LENGTH INTO PRINTNUM
	ADD	PN2,+1	
	SHIFTR	-	CONVERT LENGTH TO WORDS
		'astatus message:a'	DRIVE CTATUS NOS
		ST,0,MODE=HEX,P2=PN2	
	IF	C, IOCB, ERROR=BSCERR, T	
*	1.6	(ST+6,EQ,-1)	IF SUCCESSFUL STATUS THEN
*	PPINTE	XT 'afunction successf	
	ELSE	AT WEONCITON SUCCESSE	ELSE
		XT 'afunction failed'	2131
	ENDIF	AT BIOTOTION TRIEFS	ENDIF
TERM	EQU	*	TERMINATION POINT
	BSCCLOSE		CLOSE BSC LINE
	PROGSTOP		
*			
BSCERR	EQU	* B	SC ERROR ROUTINE
	MOVE	ST,UT	MOVE RETURN CODE
	PRINTEXT	'absc error:'	
	PRINTNUM	ST	PRINT RETURN CODE
	GOTO	TERM	GO TO TERMINATION
×			
IOCB	BSCIOCB	9,RM,0,P2=IOCB2,P3=IO	
*			IS MESSAGE ADDRESS
*		P3=10CB3	IS MESSAGE LENGTH
*			
ST	DATA		STATUS RECORD
*		10 BYTES	
*		8 BYTES	
*		1 BYTE	ETX
* *		19 BYTES	TOTAL, ROUNDED UP TO
* *		17 BYIES	10 WORDS
*			IO MOKDO

\*-- THE FOLLOWING MAY BE CHANGED FOR OTHER REQUESTS --\* × RMEQU REQUEST × RMHBSCC DATA X'1002' BSC CTRL CHARS (DLE STX) RMHID DATA C'X' HEADER ID C'R' HEADER TYPE: REQUEST **RMHTYP** DATA F'2' REQUEST TYPE: ALLOCATE RMREQ DATA RMADSN DATA CL8'MYDATA' DATA SET NAME: MYDATA VOLUME NAME: MYVOL RMAVOL DATA CL6'MYVOL' RMANREC DATA D'10' NUMBER RECORDS: 10 RMADST DATA F'1' DATA SET TYPE: DATA REQLEN EQU \*-RM LENGTH OF REQUEST **ENDPROG** END

This sample host program receives data set "MYDATA" at the remote Series/1 from the host Series/1. Data is blocked with a factor of 2, and transferred as 80-byte records.

```
EXRECV
         PROGRAM START, DS=((RECVDS, ??))
START
         EQU
         BSCOPEN
                   IOCB, ERROR=BSCEOPN
                                         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 'astatus indicates error' then print it
           PRINTNUM ST,5,MODE=HEX
                     TERM1
                                            TERMINATE
           GOTO
         ENDIF
                                         ENDIF
                   IOCB2,DT
                                         ADDRESS OF DATA
         MOVEA
         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
                   (COUNT, EQ, 0)
                                         IF FIRST TIME THEN
           BSCWRITE IX, IOCB, ERROR=BSCERR, END=BSCAB WRITE INITIAL
                                         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 'aCOUNT FAILED.
                                       COUNTED: '
           PRINTNUM COUNT
                                            PRINT COUNTS
           PRINTEXT '
                        COUNT RECORD: '
           PRINTNUM DTCCNT
         ENDIF
                                         ENDIF
```

```
EXIT POINT FOR NORMAL TERM
TERM1
         EQU
         BSCCLOSE IOCB
                                   CLOSE BSC LINE
                                   EXIT POINT FOR OPEN FAILED
TERM2
         EQU
         PROGSTOP
BSCAB
         EQU
                                       ABORT RECEIVED ON WRITE
         BSCREAD I, IOCB, ERROR = BSCERR READ STATUS
         BSCREAD
                  C, IOCB, ERROR = BSCERR READ EOT
         PRINTEXT 'DABORT RECEIVED. STATUS:'
         PRINTNUM DT,5,MODE=HEX
         GOTO
                  TERM1
                                       TERMINATE
BSCERR
                               BSC ERROR ROUTINE
         EQU
         MOVE
                  ST, EXRECV
                                       MOVE RETURN CODE
         PRINTEXT 'aBSC ERROR:'
         PRINTNUM ST
                                       PRINT RETURN CODE
         GOTO
                  TERM1
                                       GO TO TERMINATION
BSCEOPN
         EQU
                               OPEN ERROR
         MOVE
                  ST, EXRECV
                                       MOVE RETURN CODE
         PRINTEXT 'aBSC OPEN ERROR:'
         PRINTNUM ST
                                       PRINT RETURN CODE
         GOTO
                  TERM2
                                       GO TO TERMINATION
RDERR
         EQU
                               DISK READ ERROR
                  ST, EXRECV
         MOVE
                                       MOVE RETURN CODE
         PRINTEXT 'aDISK 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,O,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 TO RECEIVE A DATA SET
×
         EQU
                                REQUEST
RM
               X'1002'
RMHBSCC
         DATA
                                  BSC CNTRL CHARS (DLE STX)
               C'X'
RMHID
         DATA
                                  HEADER ID
RMHTYP
         DATA
               C'R'
                                  HEADER TYPE:
                                                    REQUEST
               F'1'
RMREQ
         DATA
                                  REQUEST TYPE:
                                                    RECEIVE
RMRDSN
         DATA
              CL8'MYDATA'
                                  DATA SET NAME:
                                                   MYDATA
RMRVOL
         DATA
              CL6'
                                  VOLUME NAME:
                                                    (IPL VOL)
         DATA
               D'0'
RMRSTR
                                  STARTING RECORD: NONE
               F'1'
                                                    SOURCE
RMRTYP
         DATA
                                  RECEIVE TYPE:
RMRBLK
         DATA
               F'2'
                                  BLOCKING FACTOR: 2
         EQU
                                LENGTH OF REQUEST
REQLEN
               *-RM
*-- STATUS RECORD
×
ST
         DATA 6F'0'
                                AREA FOR STATUS RECORD
                                 10 BYTES FOR STATUS RECORD,
×
×
                                 1 BYTE FOR ETX, ROUNDED UP
                                 TO 6 WORDS
STSFN
         EQU
               ST+6
                                STATUS FUNCTION
         EQU
                                STATUS RECORD LENGTH
STL
               *-ST
×
*-- DATA AND COUNT RECORD
×
DT
         DATA
              X'1002'
                                DATA RECORD: DLE STX
         DATA
               C'XD'
                                  HEADER ID, TYPE (DATA)
DTCCNT
                                  LOCATION OF COUNT
         EQU
               DT+10
         DATA
DTDATA
               160C' '
                                LENGTH
DTL
         EQU
               *-DT
                                DISK RECORD AREA
DISKREC
         DATA 128F'0'
         ENDPROG
         END
```

This sample host program executes a PASSTHRU session through the utility. The session is established with the Event Driven Executive supervisor. Blocking is used. All terminal I/O is performed to make the host terminal appear as if the terminal were connected at the remote Series/1.

```
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 $RMUPA IS INVOKED. AFTER THIS, A
×
         SESSION IS ESTABLISHED WITH THE EDX SUPERVISOR.
×
×
         WHENEVER A "PROGRAM END" PASSTHRU RECORD IS RECEIVED,
×
         A "REQUEST DATA" RECORD IS SENT. WHEN A "NO DATA"
         RECORD IS RECEIVED, THE OPERATOR IS ASKED WHETHER TO
×
         "ATTN" (END THE SESSION AND START ANOTHER), "READ"
×
         (TRY TO ACQUIRE DATA FROM THE HOST), OR "QUIT" (END
×
         THE PASSTHRU SESSION AND THEN TERMINATE.
×
×
START
         EQU
         BSCOPEN
                  IOCB, ERROR = BSCEOPN
                                        OPEN BSC LINE
     START UP PASSTHRU ASSIST PROGRAM ($RMUPA) IF NEEDED
*--
         QUESTION 'START PASSTHRU ASSIST PROGRAM?', NO=START2
                                         ADDRESS OF REQUEST IN IOCB
         MOVEA
                  IOCB2, REQPTAS
         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
                                         READ STATUS
         BSCREAD
                  I, IOCB, ERROR = BSCERR
         BSCREAD
                  C, IOCB, ERROR = BSCERR
                                         READ EOT
         ΙF
                   (STSFN, NE, -1)
                                         IF STATUS INDICATES ERROR
           PRINTEXT 'astatus 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
         ΙF
                   (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
                                           SET UP LENGTH IN IOCB
                     IOCB3,+RMPX
           BSCWRITE IX, IOCB, ERROR = BSCERR, END = BSCAB WRITE TO RMU
           BSCWRITE E, IOCB, ERROR = BSCERR WRITE EOT
         ELSE
                                         ELSE
           MOVE
                     ST, EXPASST
                                           SAVE RETURN CODE
           PRINTEXT 'DUNSUCCESSFUL LOAD OF PASSTHRU ASSIST PGM.'
           PRINTEXT 'aLAST MESSAGE READ:'
           PRINTNUM DT, 10, MODE = HEX
                                           PRINT MESSAGE
           PRINTEXT 'aLAST RETURN CODE FROM READ:'
           PRINTNUM ST, MODE=HEX
                                           PRINT RETURN CODE
                     TERM1
                                           TERMINATE
           GOTO
         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
                   IOCB2,ST
                                         ADDRESS OF STATUS
         MOVEA
         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 'astatus indicates error' print it
           PRINTNUM ST,5, MODE=HEX
           GOTO
                     TERM1
                                           TERMINATE
         ENDIF
                                         ENDIF
```

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

```
REQD
                                          PASSTHRU TYPE: REQ DATA
          EQU
         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
                                          SET UP PTHRU TEXT RECORD
         MOVE
                   DT,X'1002'
         MOVE
                   DT+RMPTYP,+RMPTYPTX
                                          PTHRU TYPE IS TEXT OR PFK
         MOVE
                   DT+RMPTXTL,0,BYTE
                                          ZERO HI-ORDER LENGTH BYTE
         ΙF
                   (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
                                            PLACE NUMBER IN MSG
           MOVE
                     DT+RMPTXT,DT+TXT2
           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
                   BSCST, +BSCSTO
                                          BSC STATE = RESET
         MOVE
         GOTO
                   READ
                                          END REQ TEXT PROCESSING
PGME
         EQU
                                       PASSTHRU TYPE:
                                                        PROGRAM END
                                                        (DISCONNECT)
         BSCREAD
                   C.IOCB, ERROR = BSCERR
                                          READ EOT
                                          GO AND REQUEST DATA
         GOTO
                   SNDRQD
NODA
                                          PASSTHRU TYPE: NO DATA
         EQU
         BSCREAD
                   C, IOCB, ERROR = BSCERR
                                          READ EOT
NODAQ
         PRINTEXT '0"NO DATA" RECEIVED.
                                            ENTER ONE: '
         READTEXT INMSG, 'a A(TTN), R(EAD), Q(UIT) '
         ΙF
                   (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
                                            PTHRU TYPE IS PGM END
           MOVE
                     DT+RMPTYP,+RMPTYPPE
           MOVE
                                            SET UP LENGTH IN IOCB
                     IOCB3,+RMPX
           BSCWRITE IX, IOCB, ERROR=BSCERR, END=BSCAB WRITE TO RMU
            BSCWRITE E, IOCB, ERROR = BSCERR
                                            WRITE EOT
           MOVE
                     BSCST, +BSCSTO
                                            BSC STATE = RESET
            ΙF
                     (INMSG, EQ, C'A', BYTE), GOTO, START2
                                            IF "A" THEN START NEW
                                            SESSION
            GOTO
                     TERM1
                                            OTHERWISE TERMINATE
                                          ELSE (NOT "ATTN"
         ELSE
                                               OR "QUIT")
                                                    IF "R" THEN
            ΙF
                     (INMSG, EQ, C'R'), GOTO, SNDRQD
                                                     REQUEST DATA
            GOTO
                                            ELSE ASK AGAIN
                     NODAQ
          ENDIF
                                          ENDIF
```

```
ERRPT
         EQU
                                         PASSTHRU TYPE:
                                                          UNKNOWN
         PRINTEXT 'aINVALID PASSTHRU RECORD RECEIVED:'
         PRINTNUM DT, 20, MODE = HEX
         GOTO
                  TERM1
                                         TERMINATE
         END OF CASES
*--
SNDRQD
         EQU
                                         SEND REQUEST DATA
                   DT,X'1002'
         MOVE
                                         SET UP PTHRU REQUEST DATA
         MOVE
                   DT+RMPTYP,+RMPTYPRD
                                         PTHRU TYPE IS REQEST DATA
         MOVE
                                         SET UP LENGTH IN IOCB
                   IOCB3,+RMPX
         BSCWRITE IX, IOCB, ERROR=BSCERR, END=BSCAB WRITE TO RMU
         BSCWRITE E, IOCB, ERROR = BSCERR WRITE EOT
                                         BSC STATE = RESET
         MOVE
                  BSCST, +BSCSTO
         GOTO
                   READ
                                         END REQ TEXT PROCESSING
×
TERM1
                                     EXIT POINT FOR NORMAL TERM
         EQU
         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 'DABORT RECEIVED. STATUS:'
         PRINTNUM DT, 20, MODE=HEX
         GOTO
                  TERM1
                                         TERMINATE
×
BSCERR
         EQU
                                BSC ERROR ROUTINE
               ×
         MOVE
                  ST, EXPASST
                                         MOVE RETURN CODE
         PRINTEXT 'aBSC ERROR:'
         PRINTNUM ST
                                         PRINT RETURN CODE
         GOTO
                  TERM1
                                         GO TO TERMINATION
BSCEOPN
         EQU
                                OPEN ERROR
         MOVE
                  ST, EXPASST
                                         MOVE RETURN CODE
         PRINTEXT 'aBSC OPEN ERROR:'
         PRINTNUM ST
                                         PRINT RETURN CODE
         GOTO
                  TERM2
                                         GO TO TERMINATION
```

269

```
*-- DATA AREA
×
                  LENGTH=4
                                    INPUT MSG FROM OPERATOR
INMSG
         TEXT
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
                                  REQUEST TYPE:
         DATA
               A(RMREQPST)
                                                    PASSTHRU (12)
                                  PASSTHRU BLKING
         DATA
               A(PBL)
         DATA
               H'0'
                                  FLAG (UNUSED)
         DATA
               H'0'
                                  PARTITION (UNUSED)
               CL8' '
         DATA
                                  PROGRAM:
                                                EDX SUPERVISOR
              CL6' '
         DATA
                                  VOLUME (UNUSED)
         DATA 3F'0'
                                  (REMAINDER UNUSED)
REQLEN
         EQU
               *-REQPT
                                LENGTH OF REQUEST
*-- PASSTHRU REQUEST: START PASSTHRU ASSIST PROGRAM
                                REQUEST
REQPTAS
         EQU
              X'1002'
                                  BSC CONTROL CHARS (DLE STX)
         DATA
         DATA
              C'X'
                                  HEADER ID
               C'R'
         DATA
                                  HEADER TYPE:
                                                    REQUEST
         DATA
                                  REQUEST TYPE:
                                                    PASSTHRU (12)
               A(RMREQPST)
         DATA
                                  PASSTHRU BLKING
                                                    (NONE)
               A(0)
         DATA
               H'0'
                                  FLAG (UNUSED)
         DATA
               H'0'
                                  PARTITION
                                                    (ANY)
         DATA
               CL8'$RMUPA'
                                  PROGRAM:
                                                    $RMUPA
               CL6' '
                                                    IPL
         DATA
                                  VOLUME:
         DATA
               F'0'
                                  FREE SPACE:
                                                    NONE
                                                    NONE
         DATA
               F'0'
                                  PARAMETERS:
         DATA
               F'0'
                                  DATA SETS:
                                                    NONE
               *-REQPTAS
                                LENGTH OF REQUEST
REQPTASL EQU
```

```
*-- STATUS RECORD
¥
ST
         DATA 6F'0'
                               AREA FOR STATUS RECORD
×
                                 10 BYTES FOR STATUS RECORD,
                                 1 BYTE FOR ETX, ROUNDED UP
×
¥
                                 TO 6 WORDS
STSFN
                               STATUS FUNCTION
         EQU
               ST+6
                               STATUS RECORD LENGTH
STL
         EQU
             *-ST
×
*-- 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:
         EQU
BSCSTO
              0
                                 RESET
BSCSTRD EQU
                                 READING
               1
         DATA F'0'
                               WORK WORD
N1
×
         COPY CDRRM
                               INCLUDE DEFINITION OF RMU MSGS
TXT2
         EQU
               RMPTXT+2
                               BYTE 2 OF PASSTHRU TEXT
         ENDPROG
         END
```

This sample interaction with the PASSTHRU host program illustrates running the \$DEBUG utility under the PASSTHRU function.

(Attention) > SL 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 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 > G0 OPTION(\*/ADDR/TASK/ALL): ALL 1 BREAKPOINT(S) ACTIVATED USING VOLUME EDX002 COMMAND (?): LA ZZZZ USING VOLUME EDX002 FREC SIZE 12845 FREE RECORDS IN LIBRARY COMMAND (?): \$PFO XX > WHERE INVALID COMMAND TASK AT 0274 \$ATTASK AT 2600 COMMAND (?): \$PFO XX > AT INVALID COMMAND OPTION(\*/ADDR/TASK/ALL): A BREAKPOINT ADDR: 274 LIST/NOLIST: N STOP/NOSTOP: 5 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

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

This sample host program sends data set "MYDATA" from the remote Series/1 to the host Series/1. Data is blocked with a factor of 3, and transferred as 256-byte records.

```
PROGRAM START, DS = ((SENDDS, ??))
EXSEND
START
         EQU
         BSCOPEN
                  IOCB, ERROR = BSCEOPN
                                         OPEN BSC LINE
                                         LENGTH OF REQUEST IN IOCB
         MOVE
                   IOCB3,+REQLEN
         BSCWRITE IX, IOCB, ERROR = BSCERR WRITE REQUEST
         BSCWRITE E, IOCB, ERROR = BSCERR
                                         WRITE EOT
                                         ADDRESS OF STATUS
         MOVEA
                   IOCB2,ST
         MOVE
                   IOCB3,+STL
                                         LENGTH OF STATUS IN IOCB
                                         READ STATUS
         BSCREAD I, IOCB, ERROR=BSCERR
                   (STSFN,NE,-1)
                                         IF STATUS INDICATES ERROR
           BSCREAD
                     C, IOCB, ERROR=BSCERR
                                            READ EOT
           PRINTEXT 'OSTATUS INDICATES ERROR' THEN PRINT IT
           PRINTNUM ST,5, MODE=HEX
                                           TERMINATE
           GOTO
                     TERM1
         ENDIF
                                         ENDIF
                   IOCB2,DT
         MOVEA
                                         ADDRESS OF DATA
DATA
         EQU
         MOVE
                   IOCB3,+DTL
                                         SET LENGTH TO MAX
                   C, IOCB, ERROR = BSCERR READ DATA OR COUNT
         BSCREAD
         SUB
                   IOCB, IOCB2, RESULT=RLEN COMPUTE LENGTH
         ΙF
                   (DTHTYPR, EQ, C'D', BYTE) IF DATA THEN
                                           -4 FROM LENGTH
           SUB
                     RLEN,+4
                                           FOR HEADER
           SHIFTR
                     RLEN,8
                                           RLEN = NUMBER RECORDS
                                           WRITE RECORDS NEXT
                     DS1, DTDATA, RLEN, ERROR = WRERR, END = WRERR
           WRITE
           ADD
                     COUNT, RLEN
                                           ADD NUMBER WRITTEN
                                           TO COUNT
           GOTO
                     DATA
                                           GO READ NEXT RECORD
         ELSE
                                         ELSE
                     (DTHTYPR, EQ, C'C', BYTE) IF COUNT THEN
           ΙF
                       (DTCCNT, EQ, COUNT)
                                             IF COUNT OK THEN
             ΙF
                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
                                    EXIT POINT FOR NORMAL TERM
         EQU
TERM1
         BSCCLOSE IOCB
                                    CLOSE BSC LINE
         EQU
                                    EXIT POINT FOR OPEN FAILED
TERM2
         PROGSTOP
                                BSC ERROR ROUTINE
BSCERR
         EQU
         MOVE
                  ST, EXSEND
                                        MOVE RETURN CODE
         PRINTEXT 'aBSC ERROR:'
         PRINTNUM ST
                                        PRINT RETURN CODE
                                        GO TO TERMINATION
         GOTO
                  TERM1
BSCEOPN
                                OPEN ERROR
         EQU
         MOVE
                  ST, EXSEND
                                        MOVE RETURN CODE
         PRINTEXT 'aBSC OPEN ERROR:'
         PRINTNUM ST
                                        PRINT RETURN CODE
         GOTO
                  TERM2
                                        GO TO TERMINATION
WRERR
         EQU
                                WRITE ERROR
                  ST, EXSEND
                                        MOVE RETURN CODE
         MOVE
         PRINTEXT 'aDISK 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
                                        SET UP STATUS MESSAGE
         MOVE
                  ST,X'1002'
                  ST+2,C'XS'
         MOVE
         BSCWRITE IX, IOCB, ERROR = BSCERR WRITE STATUS
         BSCWRITE E, IOCB, ERROR = BSCERR
                                        WRITE EOT
         GOTO
                  TERM1
                                        GO TO TERMINATION
IOCB
         BSCIOCB 9,RM,0,P2=IOCB2,P3=IOCB3
                                                 IOCB
×
                              P2=IOCB2 IDENTIFIES MSG ADDRESS
                              P3=IOCB3 IDENTIFIES MSG LENGTH
RLEN
                   F'0'
                                RECORD LENGTH
         DATA
         DATA
                   F'0'
                                RECORD COUNT
COUNT
*-- REQUEST TO SEND DATA SET
         EQU
RM
                                REQUEST
RMHBSCC
         DATA
               X'1002'
                                  BSC CNTRL CHARS (DLE STX)
               C'X'
RMHID
         DATA
                                  HEADER ID
                                  HEADER TYPE:
               C'R'
RMHTYP
         DATA
                                                   REQUEST
RMREQ
         DATA
              F'0'
                                  REQUEST TYPE:
                                                   SEND
              CL8'MYDATA'
                                  DATA SET NAME:
RMSDSN
         DATA
                                                   MYDATA
RMSVOL
         DATA
               CL6'
                                  VOLUME NAME:
                                                   (IPL VOL)
                                  STARTING RECORD: NONE
RMSSTR
         DATA
               D'0'
RMSTYP
         DATA
               F'0'
                                 SEND TYPE:
                                                   NORMAL
               F'3'
RMSBLK
         DATA
                                 BLOCKING FACTOR: 3
                                LENGTH OF REQUEST
REQLEN
         EQU
               *-RM
```

275

```
*-- STATUS RECORD
×
ST
        DATA 6F'0'
                              AREA FOR STATUS RECORD
                               10 BYTES FOR RECORD,
×
                               1 BYTE FOR EXT, ROUNDED
×
                               UP TO 6 WORDS
×
                              STATUS FUNCTION
STSFN
       EQU
              ST+6
STL
        EQU *-ST
                              STATUS RECORD LENGTH
×
*-- DATA AND COUNT RECORD
×
        DATA 387F'0'
                              AREA FOR DATA RECORD
DT
×
                               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
                              LENGTH
        EQU
            *-DT
        ENDPROG
        END
```

#### ERROR HANDLING

This section describes the error handling procedures of the Remote Management Utility, as well as the procedures the host program should follow upon encountering an error. The error messages displayed by the utility are also described in this section.

#### Types of Errors

As was discussed in the section "Remote Management Utility Interface" on page 207, the utility is comprised of two levels of communications protocol. Errors encountered during the transmission of these protocols by either the host or remote can be classified as follows:

- Communications errors
- Errors detected by the utility or the host program while a function is executing
- Errors detected by the utility at any time

If a communications error is encountered during a Remote Management Utility session, an error message is written to the terminal which loaded the utility. If the function requested is running when the error occurs, the function is terminated immediately by the utility. The SEND, RECEIVE, and PASSTHRU functions could however remain executing, in that these functions require multiple message exchanges between the host and the remote, before the function is completed. If the error is recoverable, the utility sends the host a Status record followed by an EOT. If necessary, an EOT ("abort") will precede the Status record. After this sequence is completed, the host may then issue a new request.

Errors detected by the utility or the host program while a function is executing include such errors as disk/diskette I/O errors during a SEND or RECEIVE operation. If the utility detects such an error, a Status record indicating the error condition is sent to the host, followed by an EOT, and the function is terminated. If necessary, an EOT ("abort") will precede the Status record. After this sequence is completed the host may issue a new request. If the host program detects an error condition, it should terminate the function in the same sequence as the utility. However, the Status record the host sends the remote requires only the 4-byte header information of a Status record (RMHBSCC, RMHID, RMHTYP fields).

Errors detected by the utility at any time include:

- Short record (text length is less than four bytes)
- Header ID (RMHID field) is not "X"
- Invalid request
- LOAD of overlay failed
- EOT not sent by the host after a request

These errors may occur any time the host sends a record to the utility. When the utility detects any of these errors, the utility sends an EOT ("abort") if necessary, followed by a Status record. The RMSST field of the Status record will contain the appropriate error code. In addition, the RMSREQ field of the Status record will contain the type of request that was in execution at the time, or a "-1" if no request was executing.

Figure 28 and Figure 29 on page 279 illustrate error handling on a SEND request.

<u>Host Program</u>	<u>Host</u>		<u>Remote</u>
(Utility sending data set)		•	
Read Continue - Data	ACK*	> <	TEXT
(At this point, Utility gets I/O error on READ to disk/diskette)			
Read Continue - Status RMHTYP='S' Status RMSREQ=0 SEND RMSFN=2 READ failed RMSST=Disk I/O return code	ACK*	> <	TEXT
Read Continue - EOT	ACK*	> <>	EOT

Figure 28. Error Handling by the Remote Management Utility

<u>Host Program</u>	<u>Host</u>		<u>Remote</u>
(Utility sending data set)		: :	
Read Continue - Data	ACK*	: > <	TEXT
(At this point, host gets error processing data record)			
Write End - Abort	EOT	>	
Write Initial - Status	ENQ	> <	ACK*
RMHBSCC DATA X'1002' RMHID DATA C'X' RMHTYP DATA C'S'	TEXT	>	
Write End - EOT	EOT	<>	ACK*

Figure 29. Error Handling by the Host Program

#### Error Messages

This section describes the error messages returned when the Remote Management Utility encounters an error. These messages are written to the terminal that loaded the utility.

#### \$RMU ERROR 1 - INSUFFICIENT BUFFER. SIZE: nnnn

The size of the buffer defined for use by the utility is less than the 512-byte minimum. The default 1024-byte buffer size has been modified incorrectly.

## \$RMU ERROR 2 - COMMUNICATIONS OPEN FAILED, RETURN CODE: nnnn

The OPEN of the BSC communications line failed. The return code is defined in the description of the BSC Access Method for the Event Driven Executive.

#### \$RMU ERROR 3 - COMMUNICATIONS CLOSE FAILED, RETURN CODE: nnnn

The CLOSE of a BSC communications line failed. The return code is defined in the description of the BSC Access Method for the Event Driven Executive.

# \$RMU ERROR 4 - COMMUNICATIONS I/O ERROR. I/O FUNCTION: aaaaaa RETURN CODE: nnnn

A communications error has been detected by the utility. The I/O function ("aaaaaa") will indicate the type of request, and is one of the following:

READ INITIAL
READ CONTINUE
WRITE EOT
WRITE INITIAL
WRITE EOT (ABORT)
WRITE CONTINUE

The return code is defined in the description of the BSC Access Method for the Event Driven Executive.

## \$RMU ERROR 5 - LOAD OVERLAY FAILED, RETURN CODE: nnnn OVERLAY NUMBER: mmmm

The utility attempted to load an overlay program via a LOAD instruction, and the load failed. The return code is defined for the LOAD instruction.

## \$RMU ERROR 6 - OVERLAY FUNCTION MISSING. FUNCTION: nnnn OVERLAY NUMBER: mmmm

The utility's function table defined a function as being contained within an overlay, but it was not. This error may occur if a user-written function is not added properly to the function table.

#### INSTALLATION

The software requirements necessary to install the Event Driven Executive Remote Management Utility on a Series/1 are distributed as part of the Event Driven Executive Version 2.0 product. The section "Hardware Requirements" on page 207 discusses the minimum hardware requirements. The host program, however, must be provided by the user.

This section describes the modules which comprise the Remote Management Utility, system generation requirements, storage requirements, and the Remote Management Utility defaults and how they can be modified.

#### Remote Management Utility Modules

The utility consists of the following modules:

\$RMU SRMUPA CDROV1 CDROV2 CDROV3 CDROV4 CDROV5 CDROVCP

CDRJP

In addition, the \$DISKUT3 utility module is required by the Remote Management Utility.

#### System Generation Requirements

The Remote Management Utility uses the Event Driven Executive BSC access method (BSCAM) and the BSC line protocol in communicating with the host system. To satisfy the BSC requirements, the BSCLINE statement must be defined at system generation. See "Chapter 3. Binary Synchronous Communications" on page 35 for details and syntax of the BSCLINE statement.

The INCLUDE statements required for binary synchronous communications are as follows:

INCLUDE BSCAM, XS2002

INCLUDE BSCINIT, XS2002

If the PASSTHRU function is to be invoked by the host program, the following INCLUDE statement is required to provide the virtual terminal support of the Event Driven Executive:

#### INCLUDE IOSVIRT, XS2002

Note: As discussed in the section "PASSTHRU Function" on page 225, the names of the virtual terminals must be CDRVTA and CDRVTB.

Refer to the <u>System Guide</u> for information on including modules at system generation.

Upon meeting the system generation requirements previously discussed, the Remote Management Utility can be loaded for execution via the \$L operator command as follows: \$L \$RMU.

#### Storage Requirements

The storage requirements for the Remote Management Utility described in this section are in addition to the storage required by the Event Driven Executive supervisor/emulator and the supervisor/emulator's required device support programs and control blocks.

The Remote Management Utility storage requirements are as follows:

- Maximum of 7K bytes plus buffer space for any function.
- The storage required by the utility can be reduced from 7K bytes to 5K bytes. If the storage is reduced to 5K bytes, all functions except ALLOCATE and DELETE can be performed with a 2K byte savings. However, when the ALLOCATE and DELETE functions are invoked, the utility will momentarily require additional storage for the \$DISKUT3 utility (which is approximately 4.5K). The storage will be obtained from the partition the Remote Management Utility is executing in.
- Storage required for loading other programs invoked through the EXEC, PASSTHRU, or SHUTDOWN functions is not considered storage required by the Remote Management Utility.

Refer to the section "Modifying Defaults" on page 283 for details on modifying storage requirements.

## Remote Management Utility Defaults

This section describes the defaults and constants within the Remote Management Utility as distributed:

- Host system ID of "HOSTRMUX"
- Remote system ID of "REMTRMUX"
- BSC device address of X'09'
- Communications line is point-to-point
- Storage required is 7K for all functions
- Buffer size is 1024 bytes

## Modifying Defaults

This section describes how the Remote Management Utility defaults can be modified to meet specific user programming requirements. The defaults can be modified via "patching" through use of the \$DISKUT2 utility. Detailed information on the \$DISKUT2 utility can be found in <a href="Utilities">Utilities</a>, Operator Commands, Program Preparation, Messages and Codes.

## Host ID

The default host ID expected by the IDCHECK function is "HOSTRMUX". This ID may be modified by applying the patch to the address illustrated in the following example, where the ID is set to "HOSTSYSA".

(Attention) > \$L \$DISKUT2

USING VOLUME EDX002

COMMAND(?): PA \$RMU \$RMU IS A PROGRAM ADDRESS: 6B6 4

(D)EC, (E)BCDIC OR (H)EX? E

NOW IS:

O6B6 C8D6 E2E3 D9D4 E4E7 HOSTRMUX

ENTER DATA: HOSTSYSA

**NEW DATA:** 

O6B6 C8D6 E2E3 E2E8 E2C1 HOSTSYSA

OK? Y

PATCH COMPLETE ANOTHER PATCH? N

COMMAND(?): EN

## Remote ID

The default remote system ID returned on a successful IDCHECK function is "REMTRMUX". This ID may be modified by applying the patch to the address illustrated in the following example, where the ID is set to "REMTSYSA".

(Attention) > \$L \$DISKUT2

USING VOLUME EDX002

COMMAND(?): PA \$RMU \$RMU IS A PROGRAM ADDRESS: 6AE 4

(D)EC, (E)BCDIC OR (H)EX? E

NOW IS:

O6AE D9C5 D4E3 D9D4 E4E7 | REMTRMUX |

ENTER DATA: REMTSYSA

NEW DATA:

O6AE D9C5 D4E3 E2E8 E2C1 | REMTSYSA |

DK? Y

PATCH COMPLETE ANOTHER PATCH? N

COMMAND(?): EN

## BSC Device Address

The default BSC device address defined in the utility is X'09'. This device address may be modified by applying the patch to the address illustrated in the following example, where the address is set to X'19'.

(Attention) > \$L \$DISKUT2 USING VOLUME EDX002 COMMAND(?): PA SRMU \$RMU IS A PROGRAM ADDRESS: 6C0 1 (D)EC, (E)BCDIC OR (H)EX? H NOW IS: 06C0 0009 1.. ENTER DATA: 0019 NEW DATA: 06C0 0019 OK? Y PATCH COMPLETE ANOTHER PATCH? N COMMAND(?): EN

#### Communications Line

The utility is distributed to run on a binary synchronous communications point-to-point communications line, either leased or switched. If the utility is to be used as a tributary station on a multipoint line (TYPE=MT on the BSCLINE statement), the patch to the address illustrated in the following example must be applied:

(Attention)
> \$L \$DISKUT2
USING VOLUME EDX002

COMMAND(?): PA \$RMU \$RMU IS A PROGRAM ADDRESS: 6D8 1

(D)EC, (E)BCDIC OR (H)EX? H

NOW IS:

06D8 0000 |.. |

ENTER DATA: 0001

**NEW DATA:** 

06D8 0001 |..

OK? Y

PATCH COMPLETE ANOTHER PATCH? N

COMMAND(?): EN

#### Storage

As was discussed in the section "Storage Requirements" on page 282, storage may be reduced from 7K to 5K. This modification can be done by applying the patch ("CDRJP") to the address illustrated in the following example.

(Attention) > \$L \$DISKUT2

USING VOLUME EDX002

COMMAND(?): PA \$RMU \$RMU IS A PROGRAM ADDRESS: 102 4

(D)EC, (E)BCDIC OR (H)EX? E

NOW IS:

ENTER DATA: CDRJP

NEW DATA:

0102 C3C4 D9D1 D740 4040 CDRJP

OK? Y
PATCH COMPLETE
ANOTHER PATCH? N

COMMAND(?): EN

## Buffer Size

The default buffer size defined in the utility is 1024 bytes. This buffer size may be modified by applying the patch to the address illustrated in the following example (nnnn is the buffer size).

(Attention) > \$L \$DISKUT2

USING VOLUME EDX002

COMMAND(?): \$5 \$RMU nnnn

OLD STORAGE SIZE WAS 1024

OK TO CONTINUE? Y

COMMAND(?): EN

Buffer sizes may be modified to allow different sizes of blocking. The following table defines maximum blocking factors and sizes for various buffer sizes:

Buffer Size	Max Blocking Factor - Standard Data Set	Max Blocking Factor - Source Data Set	Max Block Size - Passthru Data
512 (min)	1	3	248
768	2	6	504
1024 (default)	3	9	760
2048	7	22	1784
4096	15	47	3832
32512 (max)	126	403	32248

The calculations required to determine the blocking factor for the different data set types will be discussed next. The buffer size chosen should be a multiple of 256 bytes (the Event Driven Executive rounds up to the next multiple of 256 bytes).

## Standard Data Set

To determine the blocking factor for a standard data set, use the following calculation:

MSTD = (BUFF - 6) / 256

where BUFF = buffer size (bytes)
MSTD = blocking factor

Note: The remainder is discarded. The value "6" accounts for a 4-byte header, 1-byte ETX, and 1 byte for word alignment.

#### Source Data Set

To determine the blocking factor for a source data set, use the following calculation:

MSRC = (BUFF - 262) / 80

where BUFF = buffer size (bytes)
MSRC = blocking factor

<u>Note</u>: The remainder is discarded. The value "262" accounts for a 4-byte header, 1-byte ETX, 1 byte for word alignment, and 256 bytes in which the disk/diskette record is read or written.

If space is available for more than one disk/diskette record in the buffer, the utility will read or write as many records as possible at a time to increase the efficiency of disk/diskette I/O.

For example, if the buffer size is 1024 bytes and the blocking factor is 6, the utility will read or write two 256-byte records at a time.

## Passthru Data

To calculate the blocking factor for Passthru data, use the following calculation:

MPSD = BUFF - 264

where BUFF = buffer size (bytes) MPSD = Passthru data size (this is the size of the data segment of the Passthru "Text or PF Key" record)

Note: The value "264" accounts for a 6-byte header, 1-byte ETX, 1 byte for word alignment, and 256 bytes for a TEXT statement for I/O to the virtual channel.

## CDRRM Equate Listing

r						
×						ENT UTILITY
×			RI	ECORD	DESCRIP	TION
RM	EQU	0				
RMH	EQU	0	H	EADER		
RMHBSCC	EQU	0 X'100	2 '	BSC C	ONTROL	CHARS (DLE STX)
RMHID	EQU	2 C'X'		HEADE	RID	
RMHIDX	EQU	C'X'				
RMHTYP	EQU	3 CL1		HEADE	R TYPE	
RMHTYPR	EQU	C'R '		R	REQUEST	
RMHTYPS	EQU	C'S '		S	STATUS	
	EQU	c'c'		С	COUNT	
RMHTYPD		C'D'		D	DATA	
RMHTYPP		C'P'		P	PASSTHR	U
*	·-	- •		•		-
RMHX	EQU	4		EXTEN	ISION AF	TER HEADER
*						
*			RE	CORD	TYPE:	REQUEST
×						
RMREQ	EQU	RMHX+0	F	REQUE	ST TYPE	:
RMREQSND		0		0	SEND	
RMREQRCV		1		1	RECEIVE	
RMREQALC		2		2	ALLOCAT	
RMREQDEL		3		3	DELETE	_
RMREQDMP		4		4	DUMP	
RMREQURP		5		5	WRAP	
RMREQIDC		6		6	IDCHECK	
RMREQSHT		7		7	SHUTDOW	
RMREQEXC		9		9	EXEC	•
RMREQPST		12		-	PASSTHR	H
*	r 40	± £.		1.5	INJUINK	•
RMRX	EQU	RMHX+2		REQUE	ST EXTE	NSTON
*	- 40	1311117 L			. SI LAIL	
*			FY	(TENS I	nn:	SEND
*			L /		. 511 -	
RMSDSN	EQU	RMRX+0	CLS	DATA	SET NAM	F
RMSVOL	EQU					(BLANK=IPL VOLUME)
RMSSTR	EQU	RMRX+14				ORD OF DATA SET
*	r 40	KIIVATIA	U			HALF USED)
* RMSTYP	EQU	RMRX+18	<b>E</b>		OF SEND	
RMSTYPN	EQU		1.	0		(256-BYTE RECORDS,
	בעט	0		U		Y BLOCKED)
* RMSTYPS	EQU	1		1		(80-BYTE RECORDS,
	CWU	-1		1		Y BLOCKED)
* RMSBLK	EOU	DMDV±20	_	שוחמי		TOR (O OR 1=NONE)
KIJOBLK	EQU	RMRX+20	Г	DLUCK	THE LAC	IOK (O OK T-HONE)

Figure 30. CDRRM Copy Code (Part 1 of 6)

```
EXTENSION:
                                         RECEIVE
                        CL8 DATA SET NAME
RMRDSN
         EQU
               RMRX+0
                        CL6 VOLUME NAME (BLANK=IPL VOLUME)
         EQU
               RMRX+8
RMRVOL
RMRSTR
         EQU
               RMRX+14 D
                            STARTING RECORD OF DATA SET
                            (ONLY SECOND HALF USED)
RMRTYP
         EQU
               RMRX+18 F
                            TYPE OF RECEIVE
                                 NORMAL (256-BYTE RECORDS,
RMRTYPN
         EQU
                                 POSSIBLY BLOCKED)
                                 SOURCE (80-BYTE RECORDS,
RMRTYPS
         EQU
               1
                                 POSSIBLY BLOCKED)
                            BLOCKING FACTOR (O OR 1=NONE)
RMRBLK
         EQU
               RMRX+20 F
                          EXTENSION:
                                         ALLOCATE
                        CL8 DATA SET NAME
RMADSN
         EQU
               RMRX+0
         EQU
                        CL6 VOLUME NAME (BLANK=IPL VOLUME)
RMAVOL
               RMRX+8
                            NUMBER OF 256-BYTE RECORDS
RMANREC
         EQU
               RMRX+14 D
                            (ONLY SECOND HALF USED)
RMADST
         EQU
               RMRX+18 F
                            DATA SET TYPE
RMADSTU
         EQU
                              0
                                 UNDEFINED
RMADSTD
         EQU
                              1
                                 DATA
               1
                                 PROGRAM
RMADSTP
                              3
         EQU
               3
                          EXTENSION:
                                         DELETE
                        CL8 DATA SET NAME
         EQU
RMDDSN
               RMRX+0
                        CL6 VOLUME NAME (BLANK=IPL VOLUME)
RMDVOL
         EQU
               RMRX+8
                          EXTENSION:
                                         DUMP
                        CL8 DATA SET NAME
RMDPDSN
         EQU
               RMRX+0
                        CL6 VOLUME NAME (BLANK=IPL VOLUME)
RMDPVOL
         EQU
               RMRX+8
                            (UNUSED)
                        Н
RMDPPTN
         EQU
               RMRX+15 H
                            PARTITION NUMBER
                             -1
                                  REMOTE MANAGEMENT
                                  UTILITY PARTITION
                              1-8 SPECIFIC PARTITION
                          EXTENSION:
                                         WRAP
         EQU
               RMRX+0 C
                            WRAP TEXT (MAY BE ANY LENGTH)
RMWTXT
```

Figure 31. CDRRM Copy Code (Part 2 of 6)

293

```
EXTENSION:
                                          IDCHECK
RMICHK
         EQU
                RMRX+0
                        CL8 ID OF HOST
                           EXTENSION:
                                          SHUTDOWN
                        F
                             (UNUSED)
RMSDFLG
         EQU
                RMRX+2
                        Н
                             FLAG
RMSDFLGX EQU
                X'80'
                               PROGRAM TO BE EXECUTED
                X'40'
RMSDFLGL EQU
                               LOGMSG=YES
RMSDPTN
         EQU
                RMRX+3
                        Н
                             PARTITION NUMBER
                                   REMOTE MANAGEMENT
                              -1
                                   UTILITY PARTITION
×
                                   ANY PARTITION
                               1-8 SPECIFIC PARTITION
                        CL8 PROGRAM (DATA SET NAME)
RMSDPGM
         EQU
                RMRX+4
RMSDVOL
         EQU
                RMRX+12 CL6 VOLUME NAME (BLANK=IPL VOLUME)
RMSDLFS
         EQU
                RMRX+18 F
                             FREE SPACE PASSED TO PROGRAM
RMSDPRM# EQU
                RMRX+20 F
                             NUMBER OF PARAMETER WORDS
RMSDPRM
         EQU
                RMRX+22 NF
                             PARAMETER WORDS
RMSDDS#
         EQU
                RMRX+24 F
                             NUMBER OF DATA SET NAMES
                             PASSED
RMSDDS
         EQU
                RMRX+26 NF
                             DATA SET NAMES (DATA SET, VOL-
                             UME; BLANK VOLUME=IPL VOLUME)
                           EXTENSION:
                                          EXEC
                        F
                             (UNUSED)
RMXFLG
         EQU
                RMRX+2
                             FLAG
RMXFLGL
         EQU
                X'40'
                               LOGMSG=YES
                X'20'
RMXFLGW
         EQU
                               WAIT=YES
                RMRX+3
RMXPTN
         EQU
                        Н
                             PARTITION NUMBER
                              -1
                                   REMOTE MANAGEMENT
                                   UTILITY PARTITION
                                   ANY PARTITION
                               1-8 SPECIFIC PARTITION
RMXPGM
         EQU
                RMRX+4
                        CL8 PROGRAM (DATA SET NAME)
RMXVOL
         EQU
                RMRX+12 CL6 VOLUME NAME (BLANK=IPL VOLUME)
                             FREE SPACE PASSED TO PROGRAM
RMXLFS
         EQU
                RMRX+18 F
                RMRX+20 F
                             NUMBER OF PARAMETER WORDS
RMXPRM#
         EQU
         EQU
                             PARAMETER WORDS (VARIABLE)
RMXPRM
                RMRX+22 NF
                             NUMBER OF DATA SET NAMES
RMXDS#
         EQU
                RMRX+24 F
                             PASSED
RMXDS
         EQU
                RMRX+26 NF
                             DATA SET NAMES (DATA SET, VOL-
                             UME; BLANK VOLUME=IPL VOLUME)
```

Figure 32. CDRRM Copy Code (Part 3 of 6)

							······································	· · · · · · · · · · · · · · · · · · ·
*			E	XTENS	ION:	1	PASSTHRU	
*								
RMPRBLK	EQU	RMRX+0	F			FOR	RECORDS	FROM
*				REMO	TE			
×				0		NON	E	
*				OT	HER	LAR	G <b>est blo</b>	CK HOST
×						CAN	RECEIVE	
RMPRFLG	EQU	RMRX+2	Н	FLAG	CUN	USED	)	
RMPRPTN	EQU	RMRX+3	Н	PART	ITIO	N NUI	MBER	
*				-1	RE	MOTE	MANAGEM	ENT
×					UT	ILIT	Y PARTIT	ION
*				0			RTITION	
×				_			IC PARTI	TION
RMPRPGM	EQU	RMRX+4	CLS				A SET NA	
KMEKEGM *	r 40	KTKATT	0.0				A SEI NA X SUPERV	
	EOU	DMDV:12	CI.		—			
RMPRVOL	EQU							PL VOLUME
RMPRLFS	EQU	RMRX+18						PROGRAM
RMPRPRM#		RMRX+20	-				RAMETER	
RMPRPRM	EQU	RMRX+22					RDS (VAR	
RMPRDS#	EQU	RMRX+24	F	_		F DA	TA SET N	IAMES
*				PASS	ED			
RMPRDS	EQU	RMRX+26	CL8	DATA	SET	NAM	ES (DATA	SET, VOL
×			CL6	UME;	BLA	NK V	OLUME = IP	L VOLUME)
<b>*</b>								
*			R!	ECORD	TYP	E: !	STATUS	
*								
RMSREQ	EQU	RMRX+0	F	REQU	EST	TYPE		
RMSFN	EQU	RMRX+2	F	FUNC				
RMSFNOK		-1	•				EST SUCC	ESSEUL
*	240	*		•	010,	K L Q O		200102
*				1 _	20.	PEMO	TE MANAG	EMENT
* *				_			FUNCTION	
					OIIE	TIT	PONCITON	l
*	E 0.11					<b>50</b> 14	- A - I - E - B	
RMSFNID	EQU	1		_			FAILED	CW 4 1 1
RMSFNBF	EQU	2		2			REA TOO	SMALL
*		_		_		R RE		
RMSFNSHR	EQU	3		3			CORD (LE	SS THAN
×						BYTE		
RMSFNHIH	EQU	4		4	HEAD	ER I	D'IS 'H'	(INVALID
RMSFNHID	EQU	5		5	INVA	LID	HEADER I	D
<b>*</b>					(N	OT '	X' OR 'H	1)
RMSFNRQX	EQU	6		6	REQU	EST	EXPECTED	1
RMSFNREQ		7					REQUEST	
RMSFNRQS		8					SHORT (M	ISSING
*		-		_			ATION)	
RMSFNSRT	FOII	9		9				EIVE TYPE
RMSFNBLF		10					BLOCKING	
RMSFNIM	EQU	11						RECEIVED
	בעט	11		TT			REQUEST	
×					טע	KING	<b>ドログリビン</b>	

Figure 33. CDRRM Copy Code (Part 4 of 6)

RMSFNPD *	EQU	12		12		'AL 'YP		PA	SST	HRU	REC	ORD
RMSFNDPN	EQU	13		13			_	DU	MP	PAR	TITI	ON
*		•			N	UM:	BEF	?				
RMSFNRQR	EQU	14		14	REQ	UE	ST	RE	CEI	VED	WHI	LE
*										NIN		
RMSFNEOT	EQU	15		15	EOT	E	XPE	CT	ΕD	AND	NOT	
*						EC						
RMSFNVTB	EQU	16		16	VIR	TU.	AL	ΤE	RMI	NAL	BUS	Υ
*												
*				21	- 30							
*												RMSST
*					CUN	IA	TWS	K	EIU	KN	CODE	)
* RMSFNR	EOU	21		21	D F A	n ,	nte		DIC	VET	TE	
KMSFNK *	EQU	<b>C T</b>		<b>C</b> I		ΑI			ת ז ב	KET	1 5	
* RMSFNW	EQU	22		22					<b>∠</b> n τ	SKE	TTE	
*	LAO	<i>L L</i>		~ ~		AI			, ,,	JKL		
	EQU	24		24					D			
RMSFNLFP		25		25						ΔΥ	FAIL	FD
RMSFNBIO		26		26	BSC							
RMSFNVTP		27		27						ED	FOR	
*	• -									MIN		
×												
*			31	_	40:	ΕV	ENT	. D	RIV	ΈN		
*				EXE	CUTI	۷E	ΑI	DI	TIO	NAL	FUN	CTION
×					ISST							ODE
×					M \$D	IS	KUT	3	FOR	CO	DES	
*				31-	33)							
*												
RMSFNAD	-	31		31						TE	FAIL	ED
RMSFNOPN		32		32	OPE							
RMSFNSED		33		33	SET						TID	1045
RMSFNLDP *	בעט	34		34								LOAD ALID
*					1112	IK	UCI	ΙU	Cri	AKE	TIAA	WEID
l .			<b>4</b> 1	_	50:	RFI	יחא	·F	ΜΔΝ	ARF	MFNT	
*				- , ,		-						
RMSFNOFM	EQU	41		41	OVE	R L	ΑY	FU	NCT	ION	MIS	SING
*				. =	_ • •	—	•	_		•		=
RMSST	EQU	RMRX+4	F	STA	TUS	OF	F.A	IL	ING	FU	NCTI	ON
×				CCO	NTAI	NS	RE	TU	RN	COD	E IF	
*				IND	ICAT	ΕD	В	' R	MSF	(N		
RMSX	EQU	RMRX+6		STA	TUS	EX	TEN	IS I	ОN			
*												
*			EX	TEN	ISION	:	ΙI	CH	ECK	ST	ATUS	
*			<b></b>				<b>_</b>	_				
RMSRID	EQU	KMSX+0	CL8	ID	OF R	EM	UTE	: S	YST	EM		
RMSFNOFM  * RMSST  * RMSX  * *	EQU	RMRX+4	F	STA (CO IND STA	ATUS ONTAI OICAT ATUS	RL. OF NS ED EX	RRC AY FA RE BY TEN	FU IL TU R SI	NCT ING RN MSF ON ECK	FU COD (N)	MIS NCTI E IF	SING

Figure 34. CDRRM Copy Code (Part 5 of 6)

```
RECORD TYPE: COUNT
         EQU
                            REQUEST TYPE
RMCREQ
               RMHX+0
RMCFLG
         EQU
               RMHX+2
                            FLAG
RMCFLGPD EQU
               X'8000'
                            PADDING OCCURED
RMCCNT
         EQU
               RMHX+4
                        D
                            COUNT (NUMBER LOGICAL RECORDS)
               RMHX+8
                            LENGTH OF COUNT MESSAGE
         EQU
RMCL
                          RECORD TYPE:
                                       DATA
RMDDATA
         EQU
               RMHX+0 C
                            DATA (VARIABLE LENGTH)
                          RECORD TYPE:
                                       PASSTHRU
RMPTYP
         EQU
               RMHX+0
                        F
                            PASSTHRU TYPE
RMPTYPTX EQU
                                  TEXT OR PF KEY
               1
                              1
                                  REQUEST FOR DATA
RMPTYPRD EQU
               2
RMPTYPPE EQU
               3
                                  PROGRAM END (DISCONNECT)
                              3
RMPTYPND EQU
                                  NO DATA
                            PASSTHRU EXTENSION
RMPX
         EQU
               RMHX+2
                          EXTENSION:
                                        TEXT OR PF KEY
×
RMPST
         EQU
                        F
                            STATUS OF LTERM MESSAGE
               RMPX+0
RMPTXTL
         EQU
               RMPX+2
                        F
                            TEXT LENGTH (BYTES) OR -1 IF
                            PF KEY
RMPTXT
         EQU
               RMPX+4
                            TEXT (VARIABLE SIZE) IF LENGTH
                        C
                            IS NOT -1
RMPPF
                           PK KEY NUMBER (IF LENGTH IS -1)
         EQU
               RMPX+4
                            (THESE FIELDS MAY BE REPEATED
                             FOR INPUT TO HOST IF BLOCKING
                             IS REQUESTED. IF "RMPTXTL" IS
×
×
                             AN ODD NUMBER, ONE BYTE OF
                             FILLER FOLLOWS "RMPTXT".)
```

Figure 35. CDRRM Copy Code (Part 6 of 6)

#### General Description

The graphics instructions, used with the terminal support described in this book, provide a tool for the development of graphics applications. They can aid in the preparation of graphic messages, allow interactive input, and draw curves on a display terminal.

These instructions are only valid for ASCII terminals having a point-to-point vector graphics capability, and compatible with the coordinate conversion algorithm described in <u>Internal Design</u> for graphics mode control characters. The function of the various ASCII control characters used by a terminal are described in the appropriate device manual. Such terminals may be connected to the Series/1 via the #7850 Teletypewriter adapter.

Seven graphic instructions are supplied. They are used in the same manner as other instructions, except that the supporting code will be included in the user's program, rather than in the supervisor. If all instructions are coded in a program, this code requires approximately 1500 bytes of storage.

When using the instructions described in this chapter, detailed manipulation of terminal instructions and text messages are not required. All of the graphics instructions deal with ASCII data, and when sending an ASCII text string to the terminal, the XLATE=NO parameter should be coded.

Use of the graphics instructions requires that the user's object program be processed by the linkage editor program, \$LINK, in order to include the graphics functions which are supplied as object modules. Refer to the <u>Utilities</u>, <u>Operator Commands</u>, <u>Program Preparation</u>, <u>Messages and Codes</u> for the description of the autocall option of \$LINK, and for information on the use of the "AUTO=\$AUTO,ASMLIB" option of \$LINK.

The following is a list of the graphics instructions provided by the Event Driven Executive. These instructions are described in detail in the <u>Language Reference</u>.

CONCAT - Concatenate two data strings

GIN - Unscaled cursor coordinate inputs

PLOTCB - Defines graphics data area

PLOTGIN - Scaled cursor coodinate inputs

SCREEN - Converts x,y coordinates to text string

XYPLOT - Draws a x,y curve on a display YTPLOT - Plots Y points on a display

Additionally, three graphic utilities are provided. They are \$DIUTIL, \$DICOMP, and \$DIINTR. Refer to the <u>Utilities</u>, <u>Operator Commands</u>, <u>Program Preparation</u>, <u>Messages and Codes</u> for a description.

#### Hardware Considerations

Terminal support is provided for the Tektronix 4010 series of display terminals equipped the General Purpose Parallel Interface (Tektronix Custom Feature Number CM021-0109-03 with cable CM012-0541-00) or other digital I/O devices having equivalent hardware interfaces. The software provides addressing logic such that up to eight terminals may be shared on one digital input group and one digital output group, with one process interrupt bit for each terminal.

The parallel interface is intended to connect directly to the intergrated digital input/output feature (#1560). This interface consists of a driver and a receiver card, each of which has several selectable options. These options allow the user to customize the interface to his requirements. The user must refer to the manufacturer's manuals for detailed installation procedures.

The following description is intended only to supplement those manuals and guide the user when using the Event Driven Executive terminal support on the Series/1. The following Tektronix 4010 Series display terminal options should be selected:

#### Receiver Card

INTR (interrupt) PROG

ADDRESS 000(0)-111(7) to match

TERMINAL definition

PERM ADD OFF

PARITY EVEN

DELAY 3.5-18 (depends on distance)

LOGIC SENSE (3) Set all to LOW

HANDSHAKE CONTROL DATA

THRESHOLD +2 volts

MASTER OPTION None

Driver Card

LOGIC SENSE (4)
STATUS
HIGH
HANDSHAKE
HIGH
INTERRUPT
DATA
HIGH

INTERRUPT CHANNEL Use INTR

AUX TSUP OUT

ECHO OUT

PARITY EVEN, BIT 8 IN AB to A, CD to D

Before the terminal may be used with the computer, some other considerations are necessary. As noted above, the common interrupt line (INTR) should be used. It is recommended that the user select the interrupt line (0 - 7) corresponding to the terminal address. If fewer than eight terminals are attached, some of the interrupt lines will not be used. All digital input and process interrupt lines must be terminated for proper operation. If only one terminal is used, the DI terminations may have been installed by the manufacturer. With multiple terminals, all DI lines and PI lines should be terminated at the computer. A 1000-ohm resistor across the DI and PI inputs is recommended. The BAUD Rate Selection Switch should be in the "stand by" position and the J261 Connector Switch set to "interface". Both of these switches are on the Tektronix 4010 series display terminals.

When the terminal is powered on, it may be necessary to "reset" the terminal. The procedure is to put the LOCAL/LINE switch in LOCAL, back to LINE, and simultaneously press the SHIFT and RESET keys. If the terminal does not respond during normal operation, it may be necessary to perform this sequence to reset the internal circuits.

Since all input/output is done with upper case ASCII character codes, the TTY LOCK key should be activated when using the terminal with the Series/1.

The last items which merit special discussion are the GIN mode and the PAGE FULL BREAK strap options on the terminal control card (TC-2). The user must press the appropriate key followed by carriage return (CR). The PAGE FULL BREAK termination may be set to either OUT or IN, depending on the user's preference. If it is IN, the terminal will always stop when a full page condition is reached. The user must press the PAGE RESET key in order to continue. If it is OUT, the terminal will automatically go to the home address and continue printing without erasing the screen.

Decimal	Hex	Binary	EBCDIC	ASCII (see Note 1)	Eight-bit data interchange EBASC* (see Note 2)	2741 PTTC/EBCD EBCD	2741 PTTC/ Correspondence CRSP
0	00	0000 0000	NUL	NUL	NUL (even)		
1	01	0001	SOH	SOH	NUL (odd)	space	space
2	02	0010	STX	STX	@ (odd)	1	1,]
3	03	0011	ETX	ETX	@ (even)		
4	04	0100	PF	EOT	space (odd)	2	2
5	05	0101	HT	ENQ	space (even)		
6	06	0110	LC	ACK	'(even)		
7	07	0111	DEL	BEL	'(odd)	3	İ
8	08	1000		BS	DLE (odd)	4	5
9	09	1001	RLF	HT	DLE (even)		
10	0A	1010	SMM	LF	P (even)	1	
11	0B	1011	VT	VT	P (odd)	5	7
12	OC	1100	FF	FF	0 (even)	ľ	'
13	0D	1100	CR	CR	0 (odd)	6	6
13	0E	1110	SO	so	p (odd)	7	8
				SI	p (odd) p (even)	'	١
15	0F	1111 0001 0000	SI DLE	DLE	BS (odd)	8	4
16	10	1			BS (odd) BS (even)	°	4
17	11	0001	DC1	DC1	1 ' '		İ
18	12	0010	DC2	DC2	H (even)		
19	13	0011	TM	DC3	H (odd)	9	0
20	14	0100	RES	DC4	( (even)		_
21	15	0101	NL	NAK	( (odd)	0	Z
22	16	0110	BS	SYN	h (odd)	(EOA)	(EOA),9
23	17	0111	IL	ETB	h (even)		
24	18	1000	CAN	CAN	CAN (even)		
25	19	1001	EM	EM	CAN (odd)		
26	1A	1010	CC	SUB	X (even)	RS	RS
27	1B	1011	CU1	ESC	X (odd)		
28	1C	1100	IFS	FS	8 (odd)	upper case	upper case
29	1D	1101	IGS	GS	8 (even)		₹
30	1E	1110	IRS	RS	x (even)		
31	1F	1111	IUS	US	x (odd)	© (EOT)	© (EOT)
32	20	0010 0000	DS	space	EOT (odd)	@	t
33	21	0001	sos	!	EOT (even)		
34	22	0010	FS	**	D (even)		
35	23	0011	1	#	D (odd)	/	x
36	24	0100	BYP	\$	\$ (even)		
37	25	0101	LF	%	\$ (odd)	s	n
38	26	0110	ETB	&	d (odd)	t	u
39	27	0111	ESC	,	d (even)		
40	28	1000		(	DC4 (even)		
41	29	1001		<b> </b>	DC4 (odd)	u	e
42	2A	1010	SM	*	T (odd)	v	đ
43	2B	1011	CU2	+	T (even)		
44	2C	1100		Ι.	4 (even)	w	k
45	2D	1101	ENQ	'	4 (odd)	"	==
46	2E	1110	ACK	I .	t (even)		
47	2F	1111	BEL	1 ;	t (odd)	x	c
48	30	0011 0000	DEL	0	form feed (even)	] ^	"
48	31					,,	1
	1	0001	CVA	1 2	form feed (odd)	У	1 1
50	32	0010	SYN	2	L (odd)	z	h

<sup>\*</sup>The no-parity TWX code for any given character is the code that has the rightmost bit position off. The parity of the code is indicated in the parenthesis (either odd or even).

					Eight-bit		2741
			ļ		data interchange	2741	PTTC/
			ì	ASCII	EBASC*	PTTC/EBCD	Correspondence
Decimal	Hex	Binary	EBCDIC	(see Note 1)	(see Note 2)	EBCD	CRSP
51	33	0011		3	L (even)		
52	34	0100	PN	4	, (odd)		
53	35	0101	RS	5	, (even)		
54	36	0110	UC	6	l (even)	SOA	
55	37	0011 0111	EOT	7	1 (odd)	S (SOA), comma	b
56	38	1000		8	FS (odd)		
57	39	1001		9	FS (even)		
58	3A	1010		]:	\ (even)		
59	3B	1011	CU3	;	\ (odd)	index	index
60	3C	1100	DC4	<	< (even)		
61	3D	1101	NAK	=	< (odd)	B (EOB)	
62	3E	1110	GI ID	>	(odd)		
63	3F	1111	SUB	?	(even)		
64	40	0100 0000	space	@	EOA (odd)	(N) (NAK), -	!
65	41	0001		A	EOA (even)		
66	42	0010		В	B (even)		
67	43	0011	ľ	C	B (odd)	i	m
68	44	0100		D	" (even)	1.	
69	45	0101		E	" (odd)	k	
70	46 47	0110		F	b (odd)	1	v
71 72	47	0111	ĺ	G	b (even)		
73	48 49	1000		Н	DC2 (even)		
74	49 4A	1001	,	I	DC2 (odd)	m	,
75	4A 4B	1010 1011	¢	J	R (odd)	n	r
76	4B 4C	l	·	K	R (even)		
77	4C 4D	1100	<	L	2 (odd)	0	i
78	4D 4E	1101	(	M	2 (even)		
79	4E 4F	1110 1111	+	N O	r (even)		
80	50	0101 0000	]   &	O   P	r (odd)	p	a
81	51	0001	· ·		line feed (even)		
82	52	0010		Q R	line feed (odd)	q	О
83	53	0010		S	J (odd)	r	S
84	54	0100			J (even) * (odd)		
85	55	0100		T U	* (odd) * (even)		
86	56	0110		V	; (even)		
87	57	0110		w	1 ' ' '		
88	58	1000		X	; (odd) SUB (odd)	\$	w
89	59	1000		Y	SUB (even)		
90	5A	1010	!	Z	Z (even)		
91	5B	1010	\$	ا آ	Z (even) Z (odd)	CRLF	CRLF
92	5C	1100	*	'	: (even)	CKLI	CKLI
93	5D	1101	)		: (odd)	backspace	backspace
94	5E	1110	l <u>'</u>	\ \ \	z (odd)	idle	idle
95	5F	1111	<del>^</del> -	1	z (even)	, and	MIC
96	60	0110 0000	_	_	ACK (even)		
97	61	0001	1	a	ACK (odd)	&	j
98	62	0010	[ '	b	F (odd)	a	g
99	63	0011		c	F (even)	"	ь
100	64	0100		d	& (odd)	b	
101	65	0101		e	& (even)		
102	66	0110		f	f (even)		
103	67	0111		g	f (odd)	c	f
104	68	1000		h	SYN (odd)	d	
105	69	1001		i	SYN (even)	"	p
106	6A	1010	1	j	V (even)		
	6B	1011	,	k	V (odd)	e	
107							

	<del></del>	г	T	<del></del>	Eight-bit	<del></del>	2741
		į	l	l	data interchange	2741	PTTC/
				ASCII	EBASC*	PTTC/EBCD	Correspondence
Decimal	Hex	Binary	EBCDIC	(see Note 1)	(see Note 2)	EBCD	CRSP
		<del></del>	-20210	<del>                                     </del>	<u> </u>	f	
109	6D	1101		m 	6 (odd) v (odd)	1	q
110	6E 6F	1110	> ?	n	v (odd) v (even)	g	comma
111	6F 70	1111 0111 0000	l '	0	shift out (even)	h	1
112 113	70	0001	i	p	shift out (even)	11	<i>'</i>
113	72	0010		q	N (even)	]	
114	73	0010	ľ	r s	N (odd)	i	у
116	74	0100		t	. (even)	1	·
117	75	0101	[	u	. (odd)		
118	76	0110	J	v	n (odd)	(Y) (YAK), period	
119	77	0111		w	n (even)	(17th), period	
120	78	1000	ļ	x	RS (even)		
121	79	1001		у	RS (odd)		1
122	7A	1010	l :	z	↑ (odd)	horiz tab	tab
123	7B	1011	#	1 }	↑ (even)		
124	7C	1100	@	1 i	> (odd)	lower case	lower case
125	7D	1101	,	1 }	> (even)		
126	7E	1110	=	, ~	$\sim$ (even)		
127	7F	1111	,,	DEL	$\sim$ (odd)	delete	:
128	80	1000 0000			SOM (odd)		
129	81	0001	a		SOM (even)	space	space
130	82	0010	ь		A (even)	=	±, [
131	83	0011	С		A (odd)	1	
132	84	0100	d		! (even)	<	@
133	85	0101	e		! (odd)		
134	86	0110	f		a (odd)		
135	87	0111	g		a (even)	;	#
136	88	1000	h		X-ON (even)	:	%
137	89	1001	i	ļ	X-ON (odd)	1	
138	8A	1010	ļ		Q (odd)		
139	8B	1011			Q (even)	%	&
140	8C	1100	1		1 (odd)	<b>,</b>	
141	8D	1101			1 (even)		¢ *
142	8E 8F	1110	1		q (even)	>	*
143 144	90	1111 1001 0000			q (odd)	*	\$
144	91	1001 0000 0001			horiz tab (even) horiz tab (odd)	1	Ф
146	92	0010	j k		I (odd)		
146	93	0010	1		I (odd)		)
148	94	0100			) (odd)	'	,
149	95	0101	n III		) (odd)	)	z
150	96	0110	0	}	i (even)	D (EOA),"	(
151	97	0111	p		i (odd)	-	`
152	98	1000	q	Ì	EM (odd)	ì	
153	99	1001	r		EM (even)		
154	9A	1010			Y (even)	[	
155	9B	1011	ļ		Y (odd)	]	
156	9C	1100			9 (even)	upper case	upper case
157	9D	1101	ł		9 (odd)	1	İ
158	9E	1110	1		y (odd)		1
159	9F	1111	!		y (even)	C (EOT)	C (EOT)
160	<b>A</b> 0	1010 0000			WRU (even)	¢	T
161	<b>A</b> 1	0001	~		WRU (odd)		
162	A2	0010	s		E (odd)		
163	A3	0011	t		E (even)	?	X
164	A4	0100	u		% (odd)		
165	A5	0101	v	l	% (even)	S	N

		I	T T	T	Field Lis		2741
					Eight-bit	2741	2/41   PTTC/
				ACCII	data interchange EBASC*	2741 PTTC/EBCD	
Decimal	Hex	Binary	EBCDIC	ASCII (see Note 1)	(see Note 2)	EBCD	Correspondence 2741
				(see Note 1)			
166	A6	1010 0110	w		e (even)	Т	U
167	A7	0111	x		e (odd)		
168	A8	1000	У		NAK (odd)	**	_
169	A9	1001	z		NAK (even)	U	E
170	AA	1010			U (even)	V	D
171	AB	1011	1		U (odd)	***	K
172 173	AC AD	1100 1101		1	5 (even) 5 (odd)	W	K
173	AE	1110			u (odd)		
175	AF	1110			u (even)	X	С
176	B0	1011 0000			return (odd)	A	
177	В0 В1	0001			return (even)	Y	L
177	B2	0010	Į.	ļ	M (even)	Z	H
178	B3	0010		1	M (odd)	L	11
180	B4	0100			- (even)		
181	B5	0100			- (odd)		
182	B6	0110			m (odd)		
183	B7	0111	1		m (even)	(S) (SOA),	В
184	B8	1000			GS (even)	(50/1),	"
185	В9	1001		1	GS (odd)		
186	BA	1010			] (odd)		
187	BB	1011		ļ	(even)	index	index
188	BC	1100			= (odd)		
189	BD	1101			= (even)	(B) (EOB), ETB	
190	BE	1110			{ (even)		
191	BF	1111			(odd)		
192	C0	1100 0000	} }		EOM (even)	(N) (NAK),-	
193	C1	0001	A		EOM (odd)		
194	C2	0010	В		C (odd)		
195	C3	0011	C		C (even)	J	M
196	C4	0100	D		# (odd)		
197	C5	0101	E		# (even)	K	
198	C6	0110	F		c (even)	L	V
199	C7	0111	G		c (odd)		
200	C8	1000	H		X-OFF (odd)		,,,
201	C9	1001	I		X-OFF (even)	M	
202	CA	1010			S (even)	N	R
203 204	CB CC	1011	J		S (odd)		,
204	CD	1100 1101	اا		3 (even)	0	I
203	CE	1110	 	[	3 (odd) s (odd)		
200	CF	1111	'		s (odd) s (even)	P	A
207	D0	1101 0000	}		vertical tab (odd)	•	_ ^
209	D0	0001	J		vertical tab (odd)	Q	0
210	D2	0010	K	1	K (even)	R	S
211	D3	0011	L	i	K (odd)		
212	D4	0100	M		+ (even)		
213	D5	0101	N	1	+ (odd)		
214	D6	0110	o		k (odd)		
215	D7	0111	P		k (even)	!	w
216	D8	1000	Q	1	ESC (even)		
217	D9	1001	R		ESC (odd)		
218	DA	1010			[ (odd)		
219	DB	1011			[ (even)	CRLF	CRLF
220	DC	1100	1	1	;(odd)		
221	DD	1101	1		; (even)	backspace	backspace
222	DE	1110			(even)		ouchspace



Decimal	Нех	Binary	EBCDIC	ASCII (see Note 1)	Eight-bit data interchange EBASC* (see Note 2)	2741 PTTC/EBCD EBCD	2741 PTTC/ Correspondence CRSP
223	DF	1101 1111			{ (odd)		
224	EO	1110 0000	\	1	bell (odd)	1	{
225	E1	0001	,		bell (even)	+	J
226	E2	0010	S		G (even)	A	G
227	E3	0011	T		G (odd)		
228	E4	0100	U	ł	'(even)	В	+
229	E5	0101	v		'(odd)		
230	E6	0110	w		g (odd)	j	
231	E7	0111	X		g (even)	C	F
232	E8	1000	Y		ETB (even)	D	P
233	E9	1001	z		ETB (odd)	İ	
234	EA	1010			W (odd)		
235	EB	1011	l		W (even)	E	
236	EC	1100			7 (odd)		
237	ED	1101	ļ	1	7 (even)	F	l Q
238	EE	1110			w (even)	G	comma
239	EF	1111			w (odd)	İ	
240	F0	1111 0000	0	1	shift in (even)	H	?
241	F1	0001	1		shift in (odd)		
242	F2	0010	2	1	O (odd)		
243	F3	0011	3		O (even)	I	Y
244	F4	0100	4		/ (odd)		
245	F5	0101	5		/ (even)	_	
246	F6	0110	6		o (even)	$(Y)$ $(YAK)$ , $\neg$	Ì
247	F7	0111	7	1	o (odd)		l
248	F8	1000	8		US (odd)		ì
249	F9	1001	9	1	US (even)		}
250	FA	1010	LVM		← (even)	horiz tab	tab
251	FB	1011	ļ		← (odd)		
252	FC	1100			? (even)	lower case	lower case
253	FD	1101			? (odd)		
254	FE	1110	1		rub out (odd)		Ì
255	FF	1111			rub out (even)	delete	

## Notes.

- ASCII terminals attached via #7850 or #2095 with #2096.
   ASCII terminals attached via #1610 or #2091 with #2092.

#### EVENT DRIVEN EXECUTIVE LIBRARY SUMMARY

The library summary is a guide to the Event Driven Executive library. By briefly listing the content of each book and providing a suggested reading sequence for the library, it should assist you in using the library as a whole as well as direct you to the individual books you require.

## Event Driven Executive Library

The IBM Series/1 Event Driven Executive library materials consist of five full-sized books, a quick reference pocket book, and a set of tabs:

- IBM Series/1 Event Driven Executive System Guide (or System Guide), SC34-0312
- IBM Series/1 Event Driven Executive Utilities, Operator Commands, Program Preparation, Messages and Codes (or Utilities), SC34-0313
- IBM Series/1 Event Driven Executive Language Reference (or Language Reference), SC34-0314
- IBM Series/1 Event Driven Executive Communications and Terminal Application Guide (or Communications Guide), SC34-0316
- IBM Series/1 Event Driven Executive Internal Design (or Internal Design), LY34-0168
- IBM Series/1 Event Driven Executive Multiple Terminal Manager Internal Design (or Multiple Terminal Manager <u>Internal Design</u>), LY34-0190
- IBM Series/1 Event Driven Executive Indexed Access Method Internal Design (or Indexed Access Method Internal Design), LY34-0189
- IBM Series/1 Event Driven Executive Reference Summary (or Reference Summary), SX34-0101
- IBM Series/1 Event Driven Executive Tabs (or Tabs), SX34-0030

## Summary of Library

# System Guide

The <u>System Guide</u> introduces the concepts and capabilities of the Event Driven Executive system. It discusses multi-tasking, program and task structure, program overlays, storage management, and data management.

Planning aids include hardware and software requirements, along with guidelines for storage estimating.

The <u>System Guide</u> also presents step-by-step procedures for generating a supervisor tailored to your Series/1 hardware configuration and software needs.

The description of the Indexed Access Method contains the information on how to write applications that use indexed data sets.

The description of the session manager includes a procedure for modifying the session manager to include application programs in the primary option menu so that you can execute them under the session manager. You can also add a procedure to compile, link, and update programs.

Information is also provided concerning partitioned data sets, tape data organization, diagnostic aids, inter-program communication, logical screens, and dynamic data set allocation.

#### Utilities

# Utilities describes:

- Event Driven Executive utility programs
- Operator commands
- Procedures to prepare and execute system and application programs
- The session manager -- a menu-driven interface program that will invoke the programs required for program development
- Messages and codes issued by the Event Driven Executive system

The operator commands, program preparation facilities, and session manager are grouped by function and discussions include detailed syntax and explanations. The utilities are presented in alphabetical order.

## Language Reference

The <u>Language Reference</u> familiarizes you with the Event Driven Language by first grouping the instructions into functional categories. Then the instructions are listed alphabetically, with complete syntax and an explanation of each operand.

The final section of the <u>Language Reference</u> contains examples of using the Event Driven language for applications such as:

- Program loading
- User exit routine
- Graphics
- I/O level control program
- Indexing and hardware register usage

# Communications Guide

The <u>Communications Guide</u> introduces the Event Driven Executive communications support — binary synchronous communications, asynchronous communications, and the Host Communications Facility.

The <u>Communications Guide</u> contains coding details for all utilities and Event Driven language instructions needed for communications support and advanced terminal applications.

# Internal Design

Internal Design describes the internal logic flow and specifications of the Event Driven Executive system so that you can understand how the system interfaces with application programs. It familiarizes you with the design and implementation by describing the purpose, function, and operation of the various Event Driven Executive system programs.

<u>Multiple Terminal Manager Internal Design</u> and <u>Indexed Access</u> <u>Method Internal Design</u> describe the internal logic flow and specifications of these programs.

Unlike the other manuals in the library, the <u>Internal Design</u> books contain material that is the licensed property of IBM and they are available only to licensed users of the Event Driven Executive system.

## Reference Summary

The <u>Reference Summary</u> is a pocket-sized booklet to be used for quick reference. It lists the Event Driven language instructions with their syntax, the utility and program preparation commands, and the completion codes.

#### Tabs

The tabs package must be ordered separately. The package contains 33 index tabs by subject, with additional blank tabs. These extended tabular pages can be inserted at the front of various sections of the library. The tabs are color coded according to the major library topics.

### Reading Sequence

All readers of the Event Driven Executive library should begin with the first three chapters of the <u>System Guide</u> ("Introduction," "The Supervisor and Emulator," and "Data Management") for an overview of the Event Driven Executive concepts and facilities.

Readers responsible for installing and preparing the system should then continue in the <u>System Guide</u> with "System Configuration" and "System Generation."

All readers should review the <u>Utilities</u> "Introduction" to become familiar with the utility functions available for the Event Driven Executive system. Then you can read more specific sections for particular utilities, operator commands, and program preparation facilities.

After you have a basic understanding of the Event Driven Executive system and how you can best use the system for your applications, you should read the <u>Language</u> <u>Reference</u> "Introduction." This will familiarize you with the potential

of the Event Driven Language and prepare you to start coding application programs.

If you have communications support for your Event Driven Executive system, you should read the <u>Communications Guide</u>, which is an extension of the **System Guide**, **Utilities**, and the **Language** Reference.

After you know the functions of the various Event Driven Language instructions, utilities, and program preparation facilities, you may wish to refer only to the Reference Summary for correct syntax while coding your applications.

Only readers responsible for the support or modification of the Event Driven Executive system need to read Internal Design.

#### OTHER EVENT DRIVEN EXECUTIVE PROGRAMMING PUBLICATIONS

- IBM Series/1 Event Driven Executive FORTRAN IV User's Guide, SC34-0315.
- IBM Series/1 Event Driven Executive PL/I Language Reference, GC34-0147.
- IBM Series/1 Event Driven Executive PL/I User's Guide, GC34-0148.
- IBM Series/1 Event Driven Executive COBOL Programmer's Guide, SL23-0014.
- IBM Series/1 Event Driven Executive Sort/Merge Programmer's Guide, SL23-0016
- IBM Series/1 Event Driven Executive Macro Assembler Reference, GC34-0317.
- IBM Series/1 Event Driven Executive Study Guide, SR30-0436.

## OTHER SERIES/1 PROGRAMMING PUBLICATIONS

- IBM Series/1 Programming System Summary, GC34-0285.
- IBM Series/1 COBOL Language Reference, GC34-0234.
- IBM Series/1 FORTRAN IV Language Reference, GC34-0133.

- IBM Series/1 Host Communications Facility Program Description Manual, SH20-1819.
- IBM Series/1 Mathematical and Functional Subroutine Library User's Guide, SC34-0139.
- IBM Series/1 Macro Assembler Reference Summary, SX34-0128
- IBM Series/1 Data Collection Interactive Programming RPQ P82600 User's Guide, SC34-1654.

#### OTHER PROGRAMMING PUBLICATIONS

- IBM Data Processing Glossary, GC20-1699.
- IBM Series/1 Graphic Bibliography, GA34-0055.
- IBM OS/VS Basic Telecommunications Access Method (BTAM), GC27-6980.
- General Information Binary Synchronous Communications, GA27-3004.
- IBM System/370 Program Preparation Facility, SB30-1072.

## SERIES/1 SYSTEM LIBRARY PUBLICATIONS

- IBM Series/1 4952 Processor and Processor Features Description, GA34-0084.
- IBM Series/1 4953 Processor and Processor Features Description, GA34-0022.
- IBM Series/1 4955 Processor and Processor Features Description, GA34-0021.
- IBM Series/1 Communications Features Description, GA34 -0028.
- IBM Series/1 3101 Display Terminal Description, GA34-2034.
- IBM Series/1 4962 Disk Storage Unit and 4964 Diskette Unit Description, GA34-0024.
- IBM Series/1 4963 Disk Subsystem Description, GA34-0051.
- IBM Series/1 4966 Diskette Magazine Unit Description, GA34-0052.

- IBM Series/1 4969 Magnetic Tape Subsystem Description, GA34-0087.
- IBM Series/1 4973 Line Printer Description, GA34-0044.
- IBM Series/1 4974 Printer Description, GA34-0025.
- IBM Series/1 4978-1 Display Station (RPQ D02055) and Attachment (RPQ D02038) General Information, GA34-1550
- IBM Series/1 4978-1 Display Station, Keyboard (RPQ D02056) General Information, GA34-1551
- IBM Series/1 4978-1 Display Station, Keyboard (RPQ D02057) General Information, GA34-1552
- IBM Series/1 4978-1 Display Station Keyboards (RPQ D02064 and D02065) General Information, GA34-1553
- IBM Series/1 4979 Display Station Description, GA34-0026
- IBM Series/1 4982 Sensor Input/Output Unit Description, GA34-0027
- IBM Series/1 Data Collection Interactive RPQs D02312, D02313, and D02314 Custom Feature, GA34-1567

This glossary contains terms that are used in the Series/1 Event Driven Executive software publications. All software and hardware terms are Series/1 oriented. This glossary defines terms used in this library and serves as a supplement to the IBM Data Processing Glossary (GC20-1699).

\$SYSLOGA. 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.

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 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 records. Records that have been placed into an indexed data set while in load mode.

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.

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. **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.

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.

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.

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 every partition at the same address. 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.

conversion. See update.

cross partition service. A function that accesses data in two partitions.

data block. In an indexed file, an area that contains control information and data records. These blocks are a multiple of 256 bytes.

data set. A group of contiguous 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.

DDB. See disk data block.

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.

directory. A series of contiguous records in a volume that describe the contents in terms of allocated data sets and free spaces.

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.

disk data block (DDB). A control block that describes a direct access volume.

display station. An IBM 4978 or 4979 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. 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 control area (FCA). A Multiple Terminal Manager data area that describes a file access request.

file control block (FCB). In an indexed data set, the first block of the data set. It contains descriptive information about the data contained in the data set.

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.

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 as either a data block or an index block. These differ from other free blocks in that these are not initially assigned to specific logical positions in the data set.

free space. In the Indexed Access Method, record spaces or blocks that do not currently contain data, and are available for use.

free space entry (FSE). A 4-byte directory entry defining an area of free space within a volume.

FSE. See free space entry.

hardware timer. The timer features available with the Series/1 processors. Specifically, the 7840 Timer Feature card or the native timer (4952 only). Only one or the other is supported by the Event Driven Executive.

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.

index. In the Indexed Access
Method, an ordered collection of
pairs, each consisting of a key
and a pointer, used to sequence
and locate the records in an
Indexed Access Method data set.

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 data set.

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. A data set specifically created, formatted and used by the Indexed Access Method. An indexed data set may also be called an indexed file. indexed file. Synonym for indexed
data set.

index entry. In an indexed file, a key-pointer pair, where the pointer is be 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.

instruction address register (IAR). The pointer that identifies the 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).

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.

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.

key. In the Indexed Access
Method, one or more consecutive
characters in a data record, used
to identify the 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 Access Method data set, whose content is used for the key of a record.

level status block (LSB). A Series/1 hardware data area that contains processor status.

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 symbols in one or more object modules to produce another single module that is the input to the update process.

load mode. In the Indexed Access Method, the mode in which records are initially placed 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. 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.

LSB. See level status block.

mamber. 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.

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.

non-labeled 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.

overlay area. A storage area within a program reserved for overlay programs specified in the PROGRAM statement.

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.

physical timer. Synonym for hardware timer.

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-level index block. In an indexed data set, 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 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 may 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. On a display device, a field in which the operator cannot enter, modify, or erase data from the keyboard. It can contain text that the user can read.

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.

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.

recovery. The use of backup data to recreate 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 (EOI) on the reel.

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.

reorganize. For an indexed data set, the copying of the data to a new indexed data set in a manner that rearranges the data for more optimum processing and free space distribution.

return code. An indicator that reflects the results of the execution of an instruction or subroutine. The return code is placed in the task code word (at the beginning of the task control block).

roll screen. A display screen on which data is displayed 24 lines at a time or data is entered line by line, beginning with line 0 at the top of the screen and continuing through line 23 at the bottom of the screen. When a roll screen device's screen is full (all 24 lines used), an attempt to display the next line results in removal of the old screen (screen is erased) and the new line on line 0 is displayed at the top of the screen.

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 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.

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.

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.

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.

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.

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.

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 user selected options to create a supervisor tailored to the needs of a specific Series/1 configuration.

system partition. The partition that contains the supervisor (partition number 1, address space 0).

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 display station, teletypewriter or printer.

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 handl-

ing functions for terminal devices under control of the Multiple Terminal Manager.

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.

unprotected field. On a display device, a field in which the user can enter, modify, or erase data using the keyboard. Unprotected fields on a static screen are defined by the null character.

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.

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 not available for use by the system; however, it will still be available for executing I/O at the basic access level (EXIO).

vary online. To restore a device to a state where it is available for use by the system. volume. A disk or diskette subdivision defined during system configuration. A volume may contain up to 32,767 records. As many volumes may be defined for a disk as will physically fit. A diskette is limited to one volume.

volume label. A label that
uniquely identifies a single unit
of storage media.

This index is common to the Event Driven Executive library. The index includes entries from the seven publications listed below. (The Glossary is not indexed.) Each publication has a copy of the index, which provides a cross-reference between the publications.

Each page number entry contains a single letter prefix which identifies the publication where the listed subject can be found. The letter prefixes have the following meanings:

- C = Communications and Terminal Application Guide
- I = Internal Design
- L = Language Reference
- S = System Guide
- U = Utilities, Operator Commands, Program Preparation, Messages and Codes
- M = Multiple Terminal Manager Internal Design
- A = Indexed Access Method Internal Design

#### Special Characters

```
$$EDXLIB system name L-228, S-57
$$EDXVOL system name L-228, S-57
$A display active programs,
 operator command S-63, U-11
$ATTASK special task control
block L-61
$AUTO link edit auto call data
 set S-403, U-401
$B blank (clear) screen, operator command S-63, U-12
$BSCTRCE trace utility for BSC
lines C-61
$BSCUT1 trace printing utility for
      C-62
 BSC
$BSCUT2 test utility for BSC
        C-64
 lines
$C cancel a program, operator
command S-63, U-13
$COMPRES library compress $-64,
11-57
$COPY copy data sets S-64, U-59
$COPYUT1 copy data sets with allocation S-64, U-64
$CP change terminal's partition
 assignment command
    overview I-73, S-63
    syntax U-14
$D dump storage, operator command
S-63, U-15
$DASDI format disk or diskette
 S-64, U-68
$DBUGNUC debug module description
$DEBUG debugging tool U-82
$DICOMP display composer
    command description U-106
    create partitioned data set
     member S-247
    invoking U-105
overview S-67
$DIINTR display interpreter U-150
```

```
$DISKUT1 allocate/delete, list
 directory data
$JOBUTIL procedure $-229
    allocate partitioned data set
     5-248
    command descriptions U-135
    overview 5-64
$DISKUT2 patch, dump, or clear
member
    description U-142
    overview S-64
    printing I/O error logs S-275
    syntax U-143
$DISKUT3 data management utility
    description S-315
    input to S-316
    request block contents S-317
    return codes S-319, U-444
$DIUTIL display data base utility
S-248, U-150
$DUMP dump saved storage and
 registers utility U-163
$E eject printer page, operator command S-63, U-16
$EDIT1/$EDIT1N text editors
    command syntax
EDIT U-174
        EDIT mode subcommands
         U-182
        END U-175
LIST U-176
        READ
              U-177
        SUBMIT U-17 WRITE U-180
                U-179
    control keys U-172
    data set requirements
                            U-169
    line editing commands
                            U-203
    overview S-66, U-169
    summary of commands and
     subcommands U-171
$EDXASM Event Driven Language
 compiler
    features supported U-361
    internal overview I-5, I-211
    invoking
        with $JOBUTIL U-368
```

with \$L U-370	\$IAMUT1 Indexed Access Method
with session manager	utility S-148, U-235
U-369	\$IDEF \$EDXASM instruction
listing program (\$EDXLIST)	definition
U-370	description I-241
options U-358	instruction format I-226
output U-359	\$IMAGE define screen image
overlay program example I-244	utility S-68, U-250
overview S-71, U-356	usage example \$-387
programming considerations	\$IMDATA subroutine S-303
U-361	usage example \$-375
arithmetic expression	\$IMDEFN subroutine S-301
operators U-365	usage example \$-375
ATTNLIST U-365	\$IMOPEN subroutine \$-300
COPY statements U-362	usage example \$-374
ECB and QCB U-362	\$IMPROT subroutine \$-302
EQU U-365	usage example \$-375
GETEDIT and PUTEDIT U-365	\$INDEX subroutine, \$EDXASM I-233
instructions requiring	\$INITDSK initialize or verify
support modules U-365	volume S-64, U-256
IODEF statement placement	\$INITIAL automatic initialization
U-364	and restart
multiple declarations on	description S-129
DATA/DC U-363	with session manager 5-209,
source line continuation	U-28
U-361	\$IOTEST test sensor I/O, list con-
required data sets U-357	figuration S-67, U-263
usage example S-397	\$JOBUTIL job stream processor S-69, U-271
using the compiler U-356	commands U-272
\$EDXATSR supervisor interface routine I-48	
\$EDXDEF hardware configuration	set up procedure U-271 usage example S-408, U-290
editing to match hardware con-	\$L load program, operator command
figuration S-117	internals I-23
overview I-5, I-6	overview S-63
storage map I-7	syntax U-17
\$EDXL language control data set of	\$LEMSG \$LINK message data set
\$EDXASM I-221, U-357	U-401
\$EDXLIST compiler listing program	\$LINK linkage editor
U-370	data set requirements U-400
\$EDXNUC supervisor data set	description U-390
in system generation S-126	in system generation I-5
overview I-5	invoking
with \$LINK utility U-399	with \$JOBUTIL U-405
\$EDXNUC supervisor data sets	with \$L U-405
U-399	with session manager
\$EXEC language emulator linkage	U-406
I-279, I-313	overview S-71
\$EXEC session manager option	usage example \$-402
S-216, U-41	\$LNKCNTL data set S-118
\$FONT 4978 character image tables	\$LOADER 1-19, 1-22
utility S-68, U-205	module description I-78
\$FSEDIT full-screen editor, host	\$LOG I/O error logging utility
and native	description S-270, U-292
data set requirements U-209	overview S-67
options BROWSE U-213	\$LPARSE subroutine I-240
EDIT U-214	\$MOVEVOL disk volume dump/restore S-65, U-294
END U-218	\$P patch storage, operator
READ U-216	command S-63, U-18
SUBMIT U-217	\$PACK/\$UNPACK subroutines S-309
WRITE U-216	\$PDS partitioned data set utility
overview S-66, U-209	in a program S-259
primary commands U-218	overview S-65
program function (PF) keys	SPEMAP identify 4978 program
U-211	\$PFMAP identify 4978 program function keys S-68, U-301
scrolling U-210	\$PREFIND prefind data sets and
summary of options and	overlays S-69, U-302
commands U-212	\$PRT2780 spooled print utility
\$HCFUT1 Host Communications	C-72
Facility utility C-107	\$PRT3780 spooled print utility
\$IAM Indexed Access Method load	C-72
	AD 150700
module S-155 \$IAM task error exit S-178	\$RJE2780 remote job entry utility C-73, S-66

\$RJE3780 remote job entry utility C-73, S-66 \$RMU (see Remote Management Utility) \$SMCTL session manager program S-209, S-212 \$SMEND session manager program \$SMJOBR session manager program 5-212 \$SMLOG session manager program \$SMMAIN session manager program S-210, S-212, U-28 \$SMMLOG, logon menu for session manager S-212 \$SMMPRIM, primary option menu for session manager S-212, U-27, U-35 \$SMM02, program preparation secondary option menu S-214, U-37 \$SMM03, data management secondary option menu S-215, U-39 \$SMM04, terminal utilities secondary option menu S-215, U-41 \$\$MM05, graphics utilities second-ary option menu S-216, U-41 \$SMM06, execute program utilities secondary option S-216 \$SMM07, job stream processor utilities secondary option S-216 \$SMM08, communications utilities option S-217, U-43 \$SMM09, diagnostic utilities S-217, U-44 \$START supervisor entry point I - 313\$STOREMAP example I-27 \$SYSCOM data\_area I-12, I-279, I-313, S-113 \$SYSLOG system logging device overview S-110 \$SYSLOGA alternate system logging device overview S-111 \$SYSPRTR system printer overview S-111 \$S1ASM Series/1 macro assembler description U-372 internals I-5, I-253 overview S-9 storage map, general I-256 \$T set date/time, operator command S-63, U-19 \$TAPEUT1 tape management utility U-311 \$TCBCCB (ATTACH) L-59 \$TERMUT1 change terminal parameters S-68, U-334 \$TERMUT2 process 4978 image or control store S-68, U-339 restore 4974 image U-339 \$TERMUT3 send message to a terminal S-68, U-344 \$TRAP class interrupt trap utility S-67, U-348 \$UNPACK/\$PACK subroutines S-309 **\$UPDATE** object program converter description U-408 in system generation I-5 overview S-69 usage example S-407

\$UPDATEH object program converter (host) S-69, U-418
\$VARYOFF set disk, diskette, or tape offline S-63, U-20
\$VARYON set disk, diskette, or tape online S-63, U-22
with standard labeled tape S-237
\$W display date/time, operator command S-63, U-25
#1 index register 1 L-6
#2 index register 2 L-6

# Α

A after, \$FSEDIT line command U-226 A-conversion L-153 A/I (see analog input) A/O (see analog output) abort task level (SVC abend) I-49 ACCA terminal C-7, L-295 Access Method, Indexed (see Indexed Access Method) ACTION, Multiple Terminal Manager CALL coding description C-130, L-360 internals M-9 overview C-117, L-29 activate error logging, \$LOG utility U-293 realtime data member, RT \$DICOMP subcommand U-124 stopped task, GO \$DEBUG command U-93 task supervisor execution state I-43 TRAP function of storage dump, \$TRAP utility U-348 add member, \$DICOMP command U-106 advance, \$DICOMP subcommand U-111 advance X,Y (PDS) S-255 assign define key, \$TERMUT2 command U-342 add add member, AD \$DICOMP command U-106 null data set on tape volume, TA \$TAPEUT1 command U-330 options to the session manager S-224 support for new I/O terminals I-117 calling conventions I-118 code translation tables I-118 linkage conventions I-119 terminal instruction modification I-119 ADD data manipulation instruction coding description L-52 overview L-19 precision table L-53 address relocation translator I-71, S-42 addressing indexing feature L-6

ADDV data manipulation	configuring S-96
instruction coding description L-54	devices supported C-6, S-14 graphics L-26, S-46
index register use L-55	TERMINAL statement examples
overview L-19	S-106
precision table L-55	ASMERROR, \$EDXASM instruction
advance, AD \$DICOMP subcommand	I-230
U-111	assembler (see \$EDXASM)
advance and prompting input, ter- minal I/O L-46	(see \$SIASM)
AI (see analog input)	(see host assembler)
AL	assign
allocate data member, \$DIUTIL	alternate for defective 4963
command U-151	sector, \$DASDI utility U-78
allocate data set, \$DISKUT1 command U-137	DEFINE key in 4978 control store, AD \$TERMUT2 command
allocate data set, \$JOBUTIL	U-341
command U-273	asynchronous communications con-
allocate member, \$DICOMP	trol adapter (see ACCA)
command U-107	AT set breakpoints and trace
allocate data set	ranges, \$DEBUG command U-90 ATTACH task control instruction
\$JOBUTIL command U-273	coding description L-59
AL \$DISKUT1 command U-137	internals I-44
ALLOCATE function C-214	overview L-42, S-34
tape, TA \$TAPEUT1 command	attention handling, terminal I/O
U-333 member	I-108, L-47, S-63 attention keys, terminal I/O L-47
\$DICOMP command U-107	attention list (see ATTNLIST)
\$DIUTIL command U-151	ATTN key (see attention handling)
\$PDS	ATTNLIST task control statement
ALLOCATE function C-216, I-166,	\$ATTASK L-61
I-174	coding description L-61
allowable precision table L-20 alter member AL \$DICOMP command	overview L-42, S-30 attribute character, 3101 C-122
U-107	autocall
alter terminal configuration,	option, \$LINK U-401
\$TERMUT1 U-334	AUTOCALL statement requirement
alternate system logging device	(WXTRN) L-323
(\$SYSLOGA) S-47 alternate tracks S-58, U-73, U-78	automatic application initialization
ALTIAM Indexed Access Method	S-13, S-129
subroutine S-167	application restart S-13,
analog input 5-49	5-129
AI \$IOTEST command U-268	
control block I-129 IODEF statement L-187	
overview 5-49	В
SBIO instruction L-263	
SENSORIO configuration	
statement L-39	B before, \$FSEDIT line command
analog output AO \$IOTEST command U-264	U-226 backup disk or disk volume on
control block I-129	tape, ST \$TAPEUT1 command U-330
description S-49	backup dump restore utility,
IODEF statement L-186	\$MOVEVOL U-294
SBIO instruction L-264	base records, indexed data set
SENSORIO configuration statement L-39, S-84	definition S-149 loading S-160
AND data manipulation instruction	basic exchange
coding description L-57	diskette data set copy utili-
overview L-19	ty, \$COPY U-59
AO (see analog output)	basic supervisor and emulator (see
application program automatic initialization and	<pre>supervisor/emulator) batch job processing (see</pre>
restart S-129	\$JOBUTIL)
indexed access S-149	BEEP, Multiple Terminal Manager
introduction L-1	CALL
manager C-119	coding description C-137,
preparation U-351 size estimating S-344	L-361 internals M-9
structure L-8	overvieм C-117, L-29
support S-20	binary synchronous communications
ASCII terminals	automatic retry S-17
codes S-110	RSCAM/RSCAMU module

descriptions 1-80	description A-31
BSCLINE configuration state-	terminal I/O buffer
ment C-42, S-76	management I-109
control flow (BSCAM) I-147	BUFFER data definition statement
device data block (BSCDDB)	coding description L-65
I-133	overview L-17
features C-35, S-16	build data member, BU \$DIUTIL
Host Communications Facility	command U-153
protocol I-156	building an indexed data set
instruction formats C-38,	U-247
I-144	·
, ,	burst output with electronic dis-
multipoint operation C-36,	play screens L-46
S-16	bypass label processing U-311
overview S-16	description S-244
point-to-point lines S-16	
Remote Management Utility	
requirements C-208	
sample programs C-59	c
special labels for,	
description I-149	
system internal design I-133	C
	<del>-</del>
test utility, \$BSCUT2 C-64	change a key definition,
trace printing routine,	\$TERMUT2 command U-342
\$BSCUT1 C-62	copy line, \$FSEDIT line
trace routine, \$BSCTRCE C-61	command U-226
blank screen, \$B operator command	CA cancel
S-63, U-12	assembly, \$EDXASM attention
BLANK TERMCTRL function L-288	request U-358
BLDTXT subroutine, \$EDXASM I-237	copy, \$COPYUT1 attention
BLINK TERMCTRL function L-288	request U-64
BLP (see bypass label processing)	
	list option, \$FSEDIT attention
BOT (beginning-of-tape) L-40	request U-217
BOTTOM reposition line pointer,	listing, \$EDXLIST attention
\$EDIT1/N editor subcommand U-183	request U-358
boundary requirement, full-word	CAD copy all data members,
DO L-34	\$COPYUT1 command U-64
IF L-34	CALL
PROGRAM L-225	copy all members, \$COPYUT1
BP list breakpoints and trace	command U-64
ranges, \$DEBUG command U-92	program control instruction
breakpoints and trace setting, AT	coding description L-68
\$DEBUG command U-90	
	Indexed Access Method
BROWSE display data set, \$FSEDIT	syntax \$-146
option U-213	Multiple Terminal Manager
BSC (see binary synchronous	syntax L-359
communications)	overview L-32, S-31
BSCAM (see binary synchronous com-	program L-68
munications)	subroutine L-68
BSCCLOSE BSC statement I-144,	callable routines L-30
I-148	CALLFORT program control
coding description C-38	instruction
BSCDDB binary synchronous device	coding description L-70
data block	overview L-32
description of I-133	cancel
equates I-291	\$C operator command U-13
BSCEQU L-11	assembly, CA \$EDXASM attention
BSCIA immediate action routine	request U-358
(BSC) I-148	copy, CA \$COPYUT1 attention
BSCIOCB BSC statement C-39, I-144	request U-64
BSCLINE configuration statement	dump, CA \$DUMP command U-165
C-42, S-76	1 · 1 · · · · · · · · · · · · · · · · ·
BSCOPEN BSC statement C-44,	list option, CA SESEULI
	list option, CA \$FSEDIT attention request U-217
I-145, I-148	attention request U-217
	attention request U-217 listing, CA \$EDIT/N attention
BSCREAD BSC statement C-45,	attention request U-217 listing, CA \$EDIT/N attention request U-172
BSCREAD BSC statement C-45, I-145, I-148	attention request U-217 listing, CA \$EDIT/N attention request U-172 CAP copy all programs, \$COPYUT1
BSCREAD BSC statement C-45, I-145, I-148 BSCWRITE BSC statement C-49,	attention request U-217 listing, CA \$EDIT/N attention request U-172 CAP copy all programs, \$COPYUT1 command U-64
BSCREAD BSC statement C-45, I-145, I-148 BSCWRITE BSC statement C-49, I-146, I-148	attention request U-217 listing, CA \$EDIT/N attention request U-172 CAP copy all programs, \$COPYUT1 command U-64 CC copy block, \$FSEDIT line
BSCREAD BSC statement C-45, I-145, I-148 BSCWRITE BSC statement C-49, I-146, I-148 BSF (backward space file) L-75	attention request U-217 listing, CA \$EDIT/N attention request U-172 CAP copy all programs, \$COPYUT1 command U-64 CC copy block, \$FSEDIT line command U-226
BSCREAD BSC statement C-45, I-145, I-148 BSCWRITE BSC statement C-49, I-146, I-148 BSF (backward space file) L-75 BSR (backward space record) L-75	attention request U-217 listing, CA \$EDIT/N attention request U-172  CAP copy all programs, \$COPYUT1 command U-64 CC copy block, \$FSEDIT line command U-226 CCB
BSCREAD BSC statement C-45, I-145, I-148 BSCWRITE BSC statement C-49, I-146, I-148 BSF (backward space file) L-75 BSR (backward space record) L-75 BTE, buffer table entry A-20	attention request U-217 listing, CA \$EDIT/N attention request U-172  CAP copy all programs, \$COPYUT1 command U-64 CC copy block, \$FSEDIT line command U-226  CCB equate table I-292
BSCREAD BSC statement C-45, I-145, I-148 BSCWRITE BSC statement C-49, I-146, I-148 BSF (backward space file) L-75 BSR (backward space record) L-75 BTE, buffer table entry A-20 BU build data member, \$DIUTIL	attention request U-217 listing, CA \$EDIT/N attention request U-172  CAP copy all programs, \$COPYUT1 command U-64  CC copy block, \$FSEDIT line command U-226  CCB equate table I-292 internals I-105, I-119
BSCREAD BSC statement C-45, I-145, I-148 BSCWRITE BSC statement C-49, I-146, I-148 BSF (backward space file) L-75 BSR (backward space record) L-75 BTE, buffer table entry A-20 BU build data member, \$DIUTIL command U-153	attention request U-217 listing, CA \$EDIT/N attention request U-172  CAP copy all programs, \$COPYUT1 command U-64 CC copy block, \$FSEDIT line command U-226  CCB equate table I-292
BSCREAD BSC statement C-45, I-145, I-148 BSCWRITE BSC statement C-49, I-146, I-148 BSF (backward space file) L-75 BSR (backward space record) L-75 BTE, buffer table entry A-20 BU build data member, \$DIUTIL	attention request U-217 listing, CA \$EDIT/N attention request U-172  CAP copy all programs, \$COPYUT1 command U-64  CC copy block, \$FSEDIT line command U-226  CCB equate table I-292 internals I-105, I-119
BSCREAD BSC statement C-45, I-145, I-148 BSCWRITE BSC statement C-49, I-146, I-148 BSF (backward space file) L-75 BSR (backward space record) L-75 BTE, buffer table entry A-20 BU build data member, \$DIUTIL command U-153	attention request U-217 listing, CA \$EDIT/N attention request U-172  CAP copy all programs, \$COPYUT1 command U-64  CC copy block, \$FSEDIT line command U-226  CCB equate table I-292 internals I-105, I-119 interprocessor communications

CD CCREGO F-11	character constants 1-89 character image table U-205
clear data set, \$DISKUT2 com-	CHGPAN, Multiple Terminal Manager
mand U-144	CALL
copy data set, \$COPY command U-61	coding description C-135, L-364
copy data set, \$TAPEUT1	internals M-9
command U-313	overview C-124, L-29
CDATA, Multiple Terminal Manager	CL clear work data set, \$FSEDIT
CALL coding description C-139,	primary command U-221 class interrupt vector table
L-362	I-10, I-277
internals M-9	class interrupts, intercepting,
overview L-29	\$TRAP utility U-348
CDRRM equates C-292	clear data set, CD \$DISKUT2 command
CG copy all members (generic) \$COPYUT1 command U-64	U-144
CH	screen, \$B operator command
change hardcopy device,	U-12
\$BSCUT2 command C-70	CLOSE Host Communications Facili-
change host library, \$UPDATEH command U-420	ty, TP operand C-90
chain, ECB/QCB/TCB I-55	CLSRU (close tape data set) L-75 cluster, indexed data set S-200
CHAIN supervisor service routine	CM copy member
I-54	\$COPYUT1 command U-64
CHAIND supervisor service routine	\$DIUTIL command U-155
I-54 CHAINE supervisor service routine	CMDEQU L-12 CMDSETUP I-13, I-67
I-54	CNG copy all members
chaining L-27	(non-generic), \$COPYUT1 command
CHAINP supervisor service routine	U-64
I-54	CO command, \$RJE2780/\$RJE3780
change address assignment of termi-	C-76 COBOL
nal, RA \$TERMUT1 command	execution requirements S-23
U-336	link editing S-71
base address, QUALIFY \$DEBUG	overview S-7
command U-101	program preparation
character string, CHANGE \$EDIT1/N editor subcommand	requirements S-23 use with Multiple Terminal
U-184	Manager C-193
character string, change	code translation
\$FSEDIT primary command	new support tables I-111
U-219	terminal I/O layer 2 I-109
execution sequence, GOTO \$DE- BUG command U-94	code words, task L-8 COLS display columns, \$FSEDIT line
graphics or report display	command U-228
profile, \$DICOMP utility	command area, \$EDXASM I-214
U-105	command descriptions U-235
hardcopy device, CH \$BSCUT2 command C-70	COMMAND send to host, \$RJE2780/\$RJE3780 C-75
hardcopy device, RH \$TERMUT1	command table I-68, I-282, I-301
command U-338	common data area (see \$SYSCOM)
host library, CH \$UPDATEH	common emulator setup routine
command U-420 kev definition in 4978 control	command table I-13, I-282, I-301
store, C \$TERMUT2 U-342	operating conventions I-67
name of logical device, RE	communication error function
\$TERMUT1 command U-337	I-166
output volume, CV \$UPDATE	communications utilities
command U-409 page formatting parameters of	\$BSCTRCE C-61 \$BSCUT1 C-62
a terminal, CT \$TERMUT1	\$BSCUT2 C-64
U-335	\$HFCUTI C-107
partition assignment, \$CP	\$PRT2780 C-72
operator command U-14	\$PRT3780 C-72
realtime data member name RT (\$PDS) S-258	\$RJE2780 C-73 \$RJE3780 C-73
tape label support U-322	\$RMU C-282
volume	communications utilities (session
CV \$BSCUT1 command C-62	manager) S-217, U-42
CV \$COPYUT1 command U-64	communications vector table I-11,
CV \$DISKUT1 command U-137 CV \$DISKUT2 command U-143	I-278, I-313 compiler (see \$EDXASM)
CV \$UPDATEH command U-418	Compiled the Territoria

completion codes (see return	coding description L-82
codes) \$EDXASM U-436	internals I-207 overview L-18
\$IAMUT1 U-437	COPY
\$JOBUTIL U-439	block of text, CC \$FSEDIT line
\$LINK U-440	command U-226
\$UPDATE U-443	data members, all, CAD
compress	\$COPYUT1 command U-64
data base, CP \$DIUTIL command	data set, CD \$COPY command
U-154	U-61
library, \$COMPRES utility U-57	data sets with allocation, \$COPYUT1 utility U-64
compressed byte string S-309	line of text, C \$FSEDIT line
CONCAT graphics instruction	command U-226
coding description L-72	member
overview L-26	CM \$COPYUT1 command U-64
concatenating indexed data sets	CM \$DIUTIL command U-155
S-167 concurrent access L-27	all, CALL \$COPYUT1 com-
concurrent execution L-42	mand U-64
configuration statements S-75	generic, CG \$COPYUT1
configure terminal CT \$TERMUT1	command U-64
command U-335	non-generic, CNG \$COPYUT1
connecting an indexed data set	command U-64 programs, all, CAP \$COPYUT1
S-159 continuation, source program line,	command U-64
\$EDXASM U-361	text, \$EDIT1/N editor
control, device instruction level	subcommand U-186
L-24	volume, CV \$COPY command U-62
control block (see DSCB)	copy code library, instruction
control block and parameter	parsing (\$EDXASM) I-222
tables BSCEQU I-133, I-291, L-11	COPY instruction coding description L-86
CCBEQU (see also CCB) L-11	overview L-33
CMDEQU (see also emulator	Count record C-256
command table) L-12	CP compress data base, \$DIUTIL
DDBEQU I-92, I-308, L-12	command U-154
DSCBEQU (see also DSCB) L-12	CR invoke \$DISKUT1, \$IAMUT1
ERRORDEF L-12 FCBEQU A-20, L-12	command U-236 create
IAMEQU L-12	character image tables, \$FONT
PROGEQU I-312, L-13	U-205
referencing I-289	source data set, \$FSEDIT
TCBEQU (see also TCB) L-13	U-214
control block module (ASMOBJ)	supervisor for another
description I-76 CONTROL IDCB command L-175	Series/1 S-132 unique labels, \$SYSNDX
control keys for text editors	(\$EDXASM) I-242
U-172	create indexed data set S-156
control records, \$LINK U-396	cross partition instructions I-71
control statements, program	cross partition services S-286
listing L-28 task L-42	CSECT list, supervisor Version 1.1 S-347
terminal I/O forms control	Version 2 S-357
L-45	CSECT program module sectioning
CONTROL tape instruction L-74	statement
conversion	coding description L-87
algorithm for graphics I-201	overview L-33
alphameric data L-152 definition	CT change tape drive attributes,
EBFLCVT module description	\$TAPEUT1 command U-315
I-80	configure terminal, \$TERMUT1
floating point/binary I-205	command U-335
numeric data L-148	CV
program modules by \$UPDATE/H	change output volume U-409
U-418 terminal I/O binary/EBCDIC	\$UPDATE command U-409 \$UPDATEH command U-418
I-110	change volume
CONVTB data formatting	\$BSCUT1 command C-62
instruction	\$COPYUT1 command U-64
coding description L-79	\$DISKUT1 command U-137
internals I-207	\$DISKUT2 command U-143
overview L-18 CONVTD data formatting	copy volume, \$COPY command U-59
instruction	U-37
· · · = • · · · · · · · · · · · · · · ·	

CYCLE	\$DISKUT1 U-137
coding description C-132,	\$DISKUT3 S-315
L-365	\$JOBUTIL U-273
internals M-9	\$PDS S-248
overview C-116, L-29	session manager_ U-29
cylinder S-60 cylinder track sector (CTS) U-135	characteristics, HCF C-83 format
cylinder track sector (CIS) 0-135	\$FSEDIT U-210
	\$PDS S-249
	\$PRT2780 C-72
D	\$PRT3780 C-72
	naming conventions C-82, S-56
D delete line, \$FSEDIT line com-	transfer RECEIVE function C-243
mand U-228	SEND function C-247
D/I (see digital input)	utilities (see data management
D/O (see digital output)	utilities)
data	data set naming conventions, Host
conversion (see conversion)	Communications Facility C-82 data-set-shut-down condition
conversion specifications (see also conversion) L-146	S-179
definition statements L-17	date/time
files for \$S1ASM I-254	display, \$W operator command
floating-point arithmetic	U-25
instructions L-20	set, \$T operator command U-19
formatting functions L-18	DC data definition statement coding description L-88
formatting instructions L-18 integer and logical	overview L-17
instructions L-19	DCB EXIO control statement
length of transmitted, host	coding description L-91
communications I-159	overview L-24
management S-45	DCE directory control entry
management system, Indexed Access Method L-27	format I-88 DCI (Data Collection Interactive)
manipulation instructions	S-11
L-19	DD block delete, \$FSEDIT line
record contents, text editor	command U-228
I-325	DDB disk data block
representation L-20 floating-point L-20	description I-92 equate table I-308
integer L-19	DDBEQU L-12
terminal input L-45	DE delete member
terminal output L-45	\$DISKUT1 command U-137
transfer initialization,	\$DIUTIL command U-156
terminal I/O support I-112 transfer rates, Host	delete data set, \$JOBUTIL command U-274
Communications Facility C-84	deadlocks C-238, S-180
transfer ready, (DTR) BSCOPEN	debug
I-148	\$EDXASM overlay programs
Data Collection Interactive S-11	I-248
DATA data definition statement coding description L-88	aids (see also diagnostic aids) S-18
overview L-17	facility, \$DEBUG utility U-82
data management utilities	define
\$COMPRES \$-64, U-57	horizontal tabs, HTAB \$IMAGE
\$COPY S-64, U-59 \$COPYUT1 S-64, U-64	command U-252 image dimensions, DIMS \$IMAGE
\$DASDI S-64, U-68	command U-251
\$DISKUT1 S-64, U-135	indexed data set, DF \$IAMUT1
\$DISKUT1 S-64, U-135 \$DISKUT2 S-64, U-142	command U-237
\$DISKUT3 S-315	null representation, NULL
\$IAMUT1 S-148, U-235	\$IMAGE command U-253 vertical tabs, VTAB \$IMAGE
\$INITDSK S-64, U-256 \$MOVEVOL S-65, U-294	command U-254
\$PDS S-247	DEFINEQ queue processing
\$TAPEUT1 U-311	statement
session manager S-215, U-38	coding description L-94
data manipulation, vector L-19 data manipulation instructions	overview L-37 definition statements, data L-17
L-19	delete
Data record C-257	data set
data representation, terminal I/O	\$JOBUTIL command U-274
L-45	DELETE function C-216
data set	tape data set, TA \$TAPEUT1

elements, IN \$DICOMP command	digital output
U-107	digital I/O control block
member	I-129
\$PDS S-261	DO \$10TEST command U-265
DE \$DISKUT1 command U-137	external sync, XO \$IOTEST
DE \$DIUTIL command U-156	command U-266
text	IODEF statement L-186
\$EDIT1(N) editor subcom-	overview S-48
mand U-188	SBIO instruction L-267
line, D \$FSEDIT line	SENSORIO configuration
command U-228	statement L-84
with \$PREFIND U-305	DIMS define image dimensions,
DELETE function C-216, I-166,	\$IMAGE command U-251
I-174	direct access common I/O module,
DELETE instruction	DISKIO, description I-77
coding description L-329	direct access storage device
overview L-27, S-147	organization S-52
return codes L-330	direct output, DI \$DICOMP
DEQ task control instruction	subcommand U-112
coding description L-95	directory
internals I-59	control entry (DCE) I-88
overview L-42, S-33	entries S-249
supervisor function I-46	member entry (DME) I-89
DEQBSC dequeue BSC DDB routine	disaster recovery from tape, RT
I-149	\$TAPEUT1 command U-326
DEQT terminal I/O instruction	DISCONN Indexed Access Method
coding description L-97	CALL
overview L-44, S-47	coding description L-332
DETACH task control instruction	overview L-27, S-148
coding description L-98	return codes L-333
internals I-45	DISCONNECT Multiple Terminal
overview L-42, S-30	Manager utility C-119, C-159
detached, task supervisor	disconnecting an indexed data set
execution state I-43	S-159
device	DISK configuration statement S-78
busy (EXOPEN) L-129	disk/diskette
data block description, EXIO	capacity S-58
I-123	data block (DDB) I-92
instruction level control	fixed-head S-15, S-61
L-24	I/O task I-95
interrupt handling, EXIO	IPL S-16, S-61
I-125	primary volume S-60
test utility, \$IOTEST U-263	resident loading code I-19
vector table I-11, I-278	secondary volume S-60
DF define indexed file, \$IAMUT1	symbolic addressing L-10
command U-237	utilities
DI (see digital input)	\$COMPRES S-64, U-57
diagnostic	\$COPY S-64, U-59
	\$COPYUT1 S-64, U-64
aids S-265 summarized S-18	\$DASDI S-64, U-68
utilities \$DEBUG U-82	\$DISKUT1 S-64, U-135 \$DISKUT2 S-64, U-142
\$DUMP U-163	\$DISKUT3 S-315
\$10TEST U-263	\$IAMUT1 S-148, U-235
\$107E37 0-263 \$LOG U-292	\$INITDSK S-64, U-256
\$TRAP U-348	\$MOVEVOL S-65, U-294
with session manager	\$PDS S-247
S-217, U-38	utility function table U-49
	volume S-16, S-52
digital input	disk I/O instructions L-22
\$IOTEST command U-266	
digital I/O control block	DISKIO direct access common I/O
I-129	module description I-77
direct output, \$DICOMP subcom-	display (see also list)
mand U-112	character image tables, DISP
direct output to another	\$FONT command U-205
device (\$PDS) S-255	contents of storage or
display parameters, \$IAMUT1	registers, LIST \$DEBUG com-
command U-239	mand U-95
external sync, XI \$10TEST	control member (\$PDS) S-250
command U-266	control member format (\$PDS)
IODEF statement L-186	S-252
overview S-48	initial data values for image
SBIO instruction L-265	S-303
SENSORIO configuration	processor composer, \$DICOMP

processor interpreter, \$DIINTR U-150	DSR data set ready in BSCUPEN I-148
processor utility, \$DIUTIL	DTR data transfer ready in
U-150	BSCOPEN I-148
processor utility, general	DU
description U-105 profile elements (\$PDS) S-252	dump on terminal, \$DISKUT2
protected and null fields of	command U-144 dump trace file on terminal,
an image S-302	\$BSCUT1 command C-62
report line items (\$PDS)	dump
S-255	restore volume utility
status of all tasks, WHERE \$DEBUG command U-102	\$MOVEVOL U-294 storage partition, DUMP
storage, \$D operator command	function C-218
S-63, U-15	to printer
time and data, TD (\$PDS)	\$DUMP utility U-163
S-258 time and date, \$W operator	DP \$DISKUT2 command U-143 DP \$TAPEUT1 command U-317
command S-63, U-25	PR \$DICOMP command U-108
utility program set (\$PDS)	to terminal
5-248	\$DUMP utility U-163
variable, VA(\$PDS) 5-254 4978 program function keys,	DP \$TAPEUT1 command U-317 DU \$DISKUT2 command U-143
\$PFMAP utility U-301	PR \$DICOMP command U-108
DISPLAY TERMCTRL function L-288	trace file on printer, DP
DIVIDE data manipulation	\$BSCUT1 command C-62
instruction coding description L-99	trace file on terminal, DU \$BSCUT1 command C-62
overview L-19	DUMP function C-218, I-166, I-175
precision table L-100	D4969, tape device handler module
DME directory member entry	description I-82
format I-89	
updated by SETEOD S-324 DO	<del>                                      </del>
digital output (see digital	l E l
output)	<del></del>
program sequencing instruction	E-conversion (Ew.d) L-150
coding description L-101	EBFLCVT, EBDIC to floating-point
overview L-34	conversion I-205
double-precision L-19	module description I-80
floating-point arithmetic L-21	EC control echo mode, \$IAMUT1 command U-240
integer and logical L-19	ECB task control statement
DOWN move line poiner, \$EDIT1/N	coding description L-107
editor subcommand U-189 DP	internals I-55 overview L-42, S-30
dump to printer	with SBIOCB I-128
\$DISKUT2 command U-144	EDIT
\$TAPEUT1 command U-317	begin editing source data,
print trace file, \$BSCUT1 command C-62	\$EDIT1/N command U-174 create or change data set,
DR draw symbol, \$DICOMP	\$FSEDIT option U-214
subcommand U-112	enter edit mode, \$FONT
DR draw symbol (\$PDS) S-254	command U-205
draw line, LI \$DICOMP subcommand	enter edit mode, \$IMAGE command U-251
U-120	edit data set subroutine examples,
line relative LR (\$PDS) S-257	text editor I-326
symbol, DR \$DICOMP subcommand U-112	editor subcommands, \$EDIT1/N U-182
DS data set identifier, \$JOBUTIL	EDL (see Event Driven Language)
command U-275	compiler (\$EDXASM) U-356
DSCB data set control block	instruction format I-67
statement coding description L-105	<pre>interpreter, EDXALU, module   description I-77</pre>
equate table, DSCBEQU I-311	operation codes I-67
for tape, internals I-99	EDXALU Event Driven Language
internals I-92	interpreter description I-5,
overview L-22 DSCBEQU L-12	I-77 EDXFLOAT floating-point operations
DSECT (see control block and	module description I-79
parameter equate tables) L-11	EDXINIT supervisor initialization
DSOPEN subroutine	control module I-15
description S-322	description I-81

EDXLIST host listing formatter U-383	ENDSPOOL switch spool to print, \$RJE2780/\$RJE3780 C-75
EDXSTART supervisor initialization	ENDTASK task control instruction
task module description I-81	coding description L-116
EDXSVCX/EDXSVCXU task supervisor	overview L-42, S-30
addr. trans. support desc I-5,	ENQ task control instruction
I-76 EDXSYS system data tables,	coding description L-117 internals I-60
description I-75	overview L-42, S-33
EDXTIMER 7840 timer feature card	supervisor function I-45
module description I-80	ENQT terminal I/O instruction
EDXTIMR2 4952 timer module	5-293
description I-80 EDXTIO terminal I/O	coding description L-119 overview L-44, S-47
EDXTIO/EDXTIOU module	enqueue, task supervisor function
description I-78	(see ENQ)
internals I-105	entering and editing source state-
EJECT listing control statement	ments S-66, U-192
coding description L-109 overview L-28	entry points, supervisor Version 1.1 S-347
eject printer page	Version 2 S-357
\$E operator command U-16	ENTRY program module sectioning
ELSE program sequencing	statement
instruction	coding description L-121
coding description L-110, L-178	overview L-33 EOF (end-of-file) L-74
overview L-34	EOJ end of job, \$JOBUTIL command
emulator (see	U-276
supervisor/emulator)	EOP end of nested procedure,
emulator command table I-13,	\$JOBUTIL command U-276
I-282, I-301 emulator functional flow I-69	EOR data manipulation instruction coding description L-122
emulator setup routine I-67	overview L-19
command table I-13, I-282,	EOT (end-of-tape) L-41
I-301	EP end display, \$DICOMP
EN end program, \$IAMUT1 command	subcommand U-118
U-235 END	EQ (equal) L-34 EQU data definition instruction
\$LINK control record U-396	coding description L-124
option selection, \$EDXASM	overview L-17
command U-358	equate tables
option selection, \$EDXLIST	\$EDXASM compiler common area
command U-371 option selection, \$S1ASM	I-214 BSCDDB, BSC line control
U-378	block I-291
primary command input, \$FSEDIT	CCB, terminal control block
primary command U-221	I-292
task control statement coding description L-111	DDB, disk/diskette control block I-308
overview L-42	DDB for sensor I/O I-309
end display, EP \$DICOMP	DSCB, data set control block
subcommand U-118	I-311
end-of-file, indicating with	emulator command table I-282,
SETEOD S-324 ENDATIN task control instruction	I-301 Indexed Access Method A-19
coding description L-112	parameter and control block
overview L-42, S-30	L-11
ENDDO program sequencing	program header I-312
instruction	referencing I-30
coding description L-103, L-113	supervisor I-279, I-313 TCB, task control block I-314
overview L-34	ERASE terminal I/O instruction
ENDIF program sequencing	coding description L-126
instruction	overview L-44, S-47
coding description L-114,	error codes (see return codes) error handling
L-178 overview L-34	I/O error logging S-270
ENDPROG task control statement	Indexed Access Method error
coding description L-115	exit S-178
overview L-42, S-30	Remote Management Utility
ENDSEQ Indexed Access Method CALL	C-277 software trace S-265
coding description L-334 overview L-27, S-147	task error exit S-33, S-268
return codes L-335	terminal I/O L-44
	ERRORDEF L-12

ERRURS list error option	location A-20
\$EDXASM command U-358	map provided by FCBEQU S-155
\$EDXLIST command U-370	FCBEQU Indexed Access Method copy
estimating storage (see storage	code module L-12, S-155
estimating)	FDIVD dața manipulation
event control block (see ECB)	instruction
Event Driven Language (see EDL)	coding description L-137
EX exercise tape, \$TAPEUT1 com-	overview L-19
mand U-319	return codes L-138
EXEC function C-220, I-166, I-178	FETCH Host Communications
EXEC load and execute program,	Facility, TP operand C-92
\$JOBUTIL command U-277	fetch record (\$PDS) S-261
execute program	fetch status, FE \$HCFUT1 command
EXEC function C-220	C-110
PASSTHRU function C-225	file L-75
SHUTDOWN function C-251	backward space file (BSF)
utilities (session manager)	L-75
5-216	control area (see FCA)
executing, task supervisor exe-	control block (see FCB)
cution state I-43	definition L-40
exercise tape, EX \$TAPEUT1	forward space file (FSF) L-75
command U-319	manager, Multiple Terminal
EXFLIH command start I-125	Manager M-8
EXIO control instruction	tape control commands L-75
coding description L-128	FILEIO, Multiple Terminal Manager
EXIODDB device data block	CALL
description I-123	coding description C-141,
internals I-125	L-367
overview L-24, S-51	internals M-9
EXIOCLEN, EXIO termination module	overview C-118, L-29
I-126	FIND
EXIODEV configuration statement	editor commands
S-82	character string, \$EDIT1/N
EXIOINIT, system initialization	subcommand U-191
I-125	character string, \$FSEDIT
EXOPEN EXIO control instruction	primary command U-222
coding description L-129	program sequencing
internals I-125	instruction
internals 1-125	coding description L-139
	overview L-34
overview L-24 return codes L-131	FINDNOT program sequencing
external sync DI/DO, XI/XO \$IOTEST	instruction
command U-266	coding description L-141
EXTRACT, Indexed Access Method	overview L-34
CALL	FIRSTQ queue processing
coding description L-336	instruction
overview L-26, S-148	coding description L-143
return codes L-337	overview L-37, S-32
EXTRN program module sectioning	fixed-head devices S-61
statement	fixed storage area, contents I-9
coding description L-134	floating-point
overview L-33	arithmetic instruction
	equates I-283, I-303
	arithmetic instructions L-20
	binary conversions I-205
F	command entries module,
	NOFLOAT, description I-79
	operations module, EDXFLOAT,
F-conversion (Fw.d) L-149	description I-79
FADD data manipulation	return codes L-21
instruction	FMULT data manipulation
coding description L-135	instruction
overview L-19	coding description L-144
return codes L-136	overview L-19
FAN, Multiple Terminal Manager	return codes L-145
CALL	format
coding description C-139,	illustrated L-5
L-366	instruction (general) L-3
overview L-31	FORMAT data formatting statement
FCA file control area, Multiple	'A' conversion L-153
Terminal Manager C-143	'E' conversion L-150
FCB file control block for Indexed	'F' conversion L-149
Access Method	'H' conversion L-152
definition A-9, A-20	'I' conversion L-148
	coding documention 1-166

function table I-164, I-167 conversion of alphameric data L-153 conversion of numeric data L-148 G data conversion specifications L-146 L-18 module names multiple field format L-155 overview L-18 GE (greater than or equal) L-3 general instruction format L-3generating the supervisor S-115 repetitive specification L-155 GENxxxx macro I-120 GET Indexed Access Method CALL using multipliers X-type format L-154 coding description L-338 overview L-27, S-147 return codes L-340 formatted screen images S-300, U-250 formatting instructions, data GETEDIT data formatting L - 18instruction forms control coding description L-162 overview L-18 burst output with electronic **GETMAIN** storage allocation display screens L-46 forms interpretation L-46 output line buffering L-46 parameters, terminal I/O L-44 instruction Ī-25 GETPAR3 I-69 GETSEQ Indexed Access Method CALL coding description L-342 overview L-27, S-147 terminal I/O L-45 FORTRAN IV return codes L-343 GETSTORE TERMCTRL function L-288 execution requirements S-24 link editing S-71 overview S-6 GETTIME timing instruction coding description L-167
overview L-50, S-32
GETVAL subroutine, \$EDXASM I-234
GETVALUE terminal I/O instruction program preparation requirements S-24 use with Multiple Terminal Manager C-197 FPCONV data manipulation coding description L-169 overview L-44, S-47 GIN graphics instruction instruction coding description L-157 overview L-19 coding description L-172 free pool in Indexed Access overview L-26 global area, \$EDXASM I-224 GLOBAL ATTNLIST L-61 Method L-27 free space definition S-148 estimating S-168 GO activate stopped task, \$DEBUG command U-93 in Indexed Access Method L-27 GOTO free space entry I-90 change execution sequence, FREEMAIN storage allocation function I-25 \$DEBUG command U-94 coding sequencing instruction coding description L-173 FSE free space entry I-90 overview L-34 FSR (forward space record) L-75 graphics FSUB data manipulation instruction conversion algorithm I-201 functions overview L-26 coding description L-159 hardware considerations C-6, index registers L-160 overview L-19 C-300 return codes L-160 instructions L-26 CONCAT L-72 FTAB, Multiple Terminal Manager GIN L-172 CALL PLOTGIN L-210 coding description C-138, SCREEN L-270 XYPLOT L-324 L-372 XYPLOT overview C-124, L-31 return codes L-373 YTPLOT L-325 full-screen static configuration requirements L-26 terminals S-46 5-293 full-screen text editor host and native, \$FSEDIT U-209 utilities \$DICOMP U-105 full-word boundary requirement U-127 \$DIINTR DO L-34 IF L-34 \$DIUTIL U-150 session manager S-216, PROGRAM L-225 U-40 summarized S-64, U-5 function process overlays I-162 GT (greater than) L-34 function process subroutines I-162, I-170

new subroutines I-187

tape L-40, L-75 terminal S-46 timers L-50, S-32 Н I/O instructions disk L-22 H-conversion L-152 L-22 hardcopy function for terminals diskette tape L-40 PF6 I-114, U-7 IACB indexed access control block hardware levels S-30 HCF (see Host Communications built by connecting data set S-159 Facility) definition A-20 description A-35 HDR1 tape label S-239 header labels, tape S-235 header record location A-14 Remote Management Utility IAM Indexed Access Method link module S-155 C-209 IAMEQU Indexed Access Method copy code module L-12, S-155 header record format, text editor I - 323HELP list debug commands, \$DEBUG IDCB EXIO control statement command U-94 coding description L-175 higher-level index block S-197 overview L-24 horizontal tabs, defining with \$IMAGE U-252 IDCHECK function C-223, I-166, I-177 U-382 host assembler identification, verify Host Communications Facility host system C-223 C-81, I-153 IDCHECK function C-223 remote system C-223 data set naming conventions C-82 IF program sequence instruction Program Preparation coding description L-177 System/370 I-153, U-382 TPCOM module description overview L-34 I-81 II insert block, \$FSEDIT line utility program, \$HCUT1 C-107 command U-231 host program, Remote Management Utility C-205 IIB interrupt information byte, Multiple Terminal Manager C-128 host system considerations C-83 IM insert member \$DICOMP subcommand U-118 \$PDS S-257 HOSTCOMM configuration statement S-83 image dimensions, define, DIMS \$IMAGE command U-251 HX display hex words, \$DICOMP subcommand U-118 image store U-205 immediate action routines I-46 binary synchronous access I method I-149 specifying maximum number 5-88 task supervisor I-48 initialization, \$INITDSK comimmediate data L-4 mand U-257 insert line, \$FSEDIT line command U-229 initialize data base, \$DIUTIL command U-157 insert or delete elements, \$DICOMP command U-107 I-conversion (Iw) I/O device instruction level L-24 INCLUDE \$LINK control record I/O error logging data set list utility, \$DISKUT2 U-142 U-398 INCLUDE statement requirement (EXTRN) L-134 index block A-20, device table S-276 invoking S-273, U-292 log control record S-276 overview S-151 index entry A-12 index record contents, text log data set U-292 LOG macro equates S-278 syntax S-272 editor I-323 index registers printing the errors S-275 recording the errors S-270 tape log entries S-245 utility, \$LOG U-292 floating-point operations using L-21 integer operations using software introduction L-6 I/O functions indexed access control block (see IACB/ICB) disk/diskette I-95, L-22 summarized S-46 Indexed Access Method L-26, L-327 \$IAM load module S-155 \$IAMUT1 utility U-235 EXIO control I-123, L-24 summarized S-51 HOSTCOMM configuration statement L-39, S-83 overview S-45 overview S-148 parameters S-187 used in data set sensor I-127 reorganization S-166

application program

summarized S-51

	-11 C 000
preparation	cluster 5-200
\$JOBUTIL procedure S-158	data block S-194
link edit control S-158	file control block (FCB)
CALL instruction syntax L-68,	S-151, S-194
S-146	free blocks S-200
CALL processing A-4	free pool S-203
coding instructions L-327	free records \$-200
control block linkages A-15	free space S-184
control flow A-3	higher-level index block
data block location	S-197
calculation A-9	index S-195
devices supported by S-146	index block S-194
devices supported by 3 140	
diagnostic aids A-10	introduction S-151
I/O requests	last cluster 5-203
DELETE L-329, S-147	primary-level index block
DELETE L-327, 3-147	billiary rever midex prock
DISCONN L-332, S-148	(PIXB) S-152, S-195
ENDSEQ L-334, S-147	relative block number
EXTRACT L-336, S-148	(RBN) S-152
EXIKACI	(KDN) 3-132
GET L-338, S-147	reserve blocks S-201 reserve index entries
GETSEQ L-341, S-147	roserve index entries
LOAD L-344, S-147	5-202
PROCESS L-347, S-147	second-level index block
PUT L-350, S-147	(SIXB) S-152, S-197
101 L 330) 3 147	
PUTDE L-352, S-147	sequential chaining S-203
PUTUP L-354, S-147	loading and inserting records
	S-150
RELEASE L-356, S-147	
IAM link module S-155	maintenance
operation S-148	backup and recovery 5-165
overview L-27, S-145	deleting S-167
performance S-205	dumping S-167
program preparation procedure	recovery without backup
S-155	S-166
record processing A-6	reorganization S-166
request processing A-5	overview S-148
request processing a s	
request verification A-10	physical arrangement A-8
storage requirements S-204	preparing the data
indexed applications, planning and	defining the key S-166
designing	estimating free space
connecting and disconnecting	S-168
data sets S-159	selecting the block size
handling errors	S-167
data-set-shut-down condi-	putting records into S-149
tion S-179	RBN, relative block number
deadlocks S-180	A-7, A-12
error exit facilities	record locking S-146, S-160
S-178	verification A-11
long-lock-time condition	indexed data set, defining U-237
S-180	indexed file (see Indexed Access
system function return	Method)
codes S-179	indexing, address feature L-6
loading base records S-160	initial program load (see also
	IPL) I-15
processing indexed data sets	
delete S-165	initialization
direct read S-161	automatic application S-129
dinant undata C-162	disk (4962) U-68, U-73
direct update S-162	
extract S-165	disk (4963) U-68, U-78
insert S-146	diskette (4964,4966) U-68
sequential read 5-162	libraries, \$INITDSK utility
sequential update S-146	U-256
resource contention S-181	modules I-16
	modules I Io
indexed data set	nucleus I-15
base records S-149	Remote Management Utility,
building U-247	internals I-166, I-171
	THICEHOLD I IVO; I I/I
concatenating with ALTIAM	tape, \$TAPEUT1 utility U-322
subroutine S-167	task I-15
control block arrangement A-8	initialize data base, IN \$DIUTIL
	initialize data base; in Abiolit
creation with \$IAMUT1 utility	command U-157
U-236	initializing secondary volumes
formatting S-187	S-132
procedure S-156	INITMODS, initialization modules
design A-7	I-16
	INITTASK, initialization task
determining size and format	
U-247	I-15
format	input, terminal I/O L-46
blocks S-192	triplicate and minima as of the 10

Input Buffer, Multiple Terminal Manager C-116	IPL
contents during 4978/4979/3101	
buffer operation C-129	
description C-116	
input data parsing, description	
of I-218 Input Error function I-166, I-182	
input/output (see I/O)	
input output control block (see	
IOCB)	
INPUT switch to input mode, \$EDIT1/N editor subcommand U-192	
insert	
block, II \$FSEDIT line com-	
mand U-231	
elements, IN \$DICOMP command	IPLS
U-107	Mar
line, I \$FSEDIT line command U-229	
member, IM \$DICOMP subcommand	1
U-118	
instruction address register (see	,
IAR)	
instruction and statements - over- view L-15	job job
instruction definition and	JOB
checking (\$EDXASM) I-241	con
instruction format, Event Driven	job
Language I-67, L-3	, S - 6
instruction format, general L-3 instruction operands L-3	job (se
integer and logical instructions	JP
L-19	٠.
interactive program debugging	
\$-67, U-82	ın.
interface routines, supervisor I-61	JR j sub
interprocessor communications	JUMF
C-29	jump
interprogram dialogue 5-282	sub
interrupt, from EXIO device I-125	jump
interrupt information byte (see IIB)	sub
interrupt line S-313	
interrupt servicing I-46, I-113	1
INTIME timing instruction	
coding description_L-181	•
overview L-50, S-32 introduction to EDL L-1	key
invoking the loader I-23	key
invoking the session manager U-27	keyb
invoking the utilities U-47	min
IOCB terminal I/O instruction	keyb
coding description L-183 constructing, for formatted	STE KEYS
screen (\$IMDEFN) S-301	1(21)
overview L-44, S-47	keyu
structure S-296	
terminal I/O instruction	
L-183 TERMINAL statement converted	
to S-96	t
IODEF sensor based I/O statement	
U-364	LA
coding description L-185	
overview L-39, S-51 SPECPI - process interrupt	
user routine L-189	
IOLOADER, function of I-127	
IOLOADER/IOLOADRU sensor based I/O	
init. module desc. I-78	labe
IOR data manipulation instruction	
coding description L-191 overview L-19	

automatic application initialization and restart S-129 messages U-421 date and time U-425 IPL operation U-421 load utility location U-424 sensor I/O status check U-424 storage map generation U-423 tape initialization U-423 volume initialization U-422 procedure U-421 SCRN, Multiple Terminal nager C-125

J

job U-278
job control statement U-278
JOB job identifier, \$JOBUTIL
command U-278
job stream processor, \$JOBUTIL
S-69, U-271
job stream processor utilities
(session manager) S-216
JP
jump (\$PDS) S-255
to address, \$DICOMP
subcommand U-118
JR jump reference, \$DICOMP
subcommand U-118
JUMP, \$JOBUTIL command U-279
jump reference, JR \$DICOMP
subcommand U-118
jump to address, JP \$DICOMP
subcommand U-118

K

key (see program function (PF) keys keyboard and ATTNLIST tasks, terminal I/O L-47 keyboard define utility for 4978, \$TERMUT2 U-339 KEYS list program function keys \$IMAGE command U-253 keyword operand L-5

L

LA

display directory, \$DIUTIL
command U-158
list all members, \$DISKUT1
command U-135, U-136
list terminal assignment,
\$TERMUT1 command U-336
label L-3
field L-3
syntax description L-4

LABEL end jump, \$JOBUTIL command	U-407
U-280	output from \$LINK U-403
labels, tape (see tape) LABELS subroutine, \$EDXASM I-238	storage map U-393 link edit program object modules
LACTS list all members CTS mode,	U-390
\$DISKUT1 command U-135	link module, Indexed Access
language control data set,	Method S-155
\$EDXASM I-221, U-357	linkage editor S-71, U-353
LASTQ queue processing	LINKON, Multiple Terminal Manager
instruction	CALL
coding description L-191	coding description C-132,
overview L-37, S-32	L-376 internals M-9
layers, terminal I/O I-108 LB display characters	overview C-115, L-29
\$DICOMP display character sub-	
command U-119	active programs, \$A operator
\$PDS S-252	command U-11
LC load control store, \$TERMUT2	breakpoints and trace ranges,
command U-342	BP \$DEBUG command U-92
LD	characters, LB \$DICOMP
list all hardware devices,	subcommand U-119
\$IOTEST command U-269 list data members, \$DISKUT1	data members, LD \$DISKUT1 command U-138
command U-138	data members, LDCTS \$DISKUT1
LDCTS list data members CTS mode,	command U-135
\$DISKUT1 command U-135	data set
LE (less than or equal) L-34	BROWSE \$FSEDIT option
level status block (see LSB)	U-213
LEWORK1 work data set for \$LINK	LP \$DISKUT2 command U-143
U-400	LU \$DISKUT2 command U-146
LEWORK2 work data set for \$LINK	status, ST \$DIUTIL
U-400	command U-162
LH display member header, \$DIUTIL command U-159	date∕time, \$W operator command U-25
LI	date/time, TD \$DICOMP
draw line \$DICOMP subcommand	subcommand U-124
U-120	devices, LD \$IOTEST command
draw line \$PDS S-253	U-269
load image store, \$TERMUT2	end, EP \$DICOMP subcommand
command U-342	U-117
library definition S-52	error specification, ERRORS
directory, disk or diskette	\$EDXASM command U-358 hardware configuration, LD
I-87	\$IOTEST command U-264
origin S-60	insert mask, MASK \$FSEDIT line
line	command U-232
commands, \$FSEDIT U-229	member, LM \$DISKUT1 command
continuation, source	U-138
statement L-4	member, PR \$DICOMP command
editing, \$EDIT1/N U-203	U-108 member header, LH \$DIUTIL com-
pointer reposition (see move line pointer)	mand U-159
source line continuation	members, all
U-361	LA \$DISKUT1 command U-135
LINK, Multiple Terminal Manager	LA \$DIUTIL command U-158
CALL	LACTS \$DISKUT1 command
coding description C-131,	U-135
L-374	processor program, \$EDXLIST
internals M-9	U-370
overview C-115, L-29 link edit process, \$LINK U-394	program function key codes, \$PFMAP utility U-301
autocall option U-393	program function keys, KEYS
building an EDX supervisor	\$IMAGE command U-253
U-394	program members, LP \$DISKUT1
combining program modules	command U-139
U-392	program members, LPCTS
control records U-396	\$DISKUT1 command U-135
elimination of duplication	status of all tasks, WHERE
control sections U-393	\$DEBUG command U-102
formatting modules for	storage, \$D operator command
\$UPDATE U-392 input to \$LINK U-396	U-15 terminal
multiple control sections	names/types/addresses, LA
U-392	\$TERMUT1 command U-335
object module record format	variables. VA \$DICOMP

subcommand U-125	location dictionary I-250
volume information, VI \$IOTEST	lock
command U-270	locks, block and record A-16
LIST commands	locks, file A-17
data set	record S-146
LIST \$EDIT1/N command	LOCK TERMCTRL function L-288
U-193	LOG
LIST \$FSEDIT option U-217	I/O error logging macro S-271
display lines of text,	job processor commands,
\$EDIT1/N editor subcommand	\$JOBUTIL command U-281
U-193	log data set for I/O errors U-292
display storage or registers,	logical end-of-file on disk S-324
\$DEBUG command U-95	logical screens S-293
lines of text, LIST \$EDIT1/N	logon menu for session manager
editor command U-176	S-212, U-27
list device option, \$EDXASM	long-lock-time condition S-180
command U-358	low storage
list device option, \$EDXLIST	during IPL I-16
command U-370	during program load I-20
obtain full listing, LIST	LP
\$EDXASM command U-358	list data set on printer,
print data set, \$EDIT1/N	\$DISKUT2 command U-144
command U-176	list program members, \$DISKUT1
print data set, \$FSEDIT	command U-139
option U-217	LPCTS list program members CTS
registers, LIST \$DEBUG	mode, \$DISKUT1 command U-135
command U-95	LR draw line relative
storage, LIST \$DEBUG command	\$DICOMP subcommand U-121
U-95	\$PDS S-257
listing control functions U-29	LS
listing control instructions	list space, \$DISKUT1 command
EJECT L-109	U-140
overview L-28	list supervisor configuration,
PRINT L-216	\$IOTEST command U-270
SPACE L-275	LSB level status block I-52,
	U-427
TITLE L-308	
LISTP list to \$SYSPRTR, \$DISKUT1	LT (less than) L-34
command U-135	LU list data set on console,
LISTT list to terminal, \$DISKUT1	\$DISKUT2 command U-146
command U-135	LV list through volumes, \$DISKUT1
LL list log data set, \$DISKUT2	U-141
command U-145	
LM list member, \$DISKUT1 command	
U-138	[]
U-138 LO load indexed file, \$IAMUT1	М
U-138 LO load indexed file, \$IAMUT1 command U-241	М
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD	<b></b>
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD Indexed Access Method CALL	M move line, \$FSEDIT line command
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD Indexed Access Method CALL coding description L-344	<b></b>
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD Indexed Access Method CALL coding description L-344 connect file S-159	M move line, \$FSEDIT line command U-233 macro assembler
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD Indexed Access Method CALL coding description L-344 connect file S-159	M move line, \$FSEDIT line command U-233
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346	M move line, \$FSEDIT line command U-233 macro assembler
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146	M move line, \$FSEDIT line command U-233 macro assembler internal overview \$S1ASM
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD  Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346 task control instruction coding description L-194	M move line, \$FSEDIT line command U-233 macro assembler internal overview \$S1ASM I-253
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD  Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346 task control instruction	M move line, \$FSEDIT line command U-233 macro assembler internal overview \$S1ASM I-253 overview S-9 macro library S-6
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD  Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346 task control instruction coding description L-194	M move line, \$FSEDIT line command U-233 macro assembler internal overview \$S1ASM I-253 overview S-9
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD  Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346 task control instruction coding description L-194 internals I-24	M move line, \$FSEDIT line command U-233 macro assembler internal overview \$S1ASM I-253 overview S-9 macro library S-6 macro library/host S-5 magazine diskette (see 4966
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD  Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346 task control instruction coding description L-194 internals I-24 overview L-42	M move line, \$FSEDIT line command U-233 macro assembler internal overview \$S1ASM I-253 overview S-9 macro library S-6 macro library/host S-5 magazine diskette (see 4966 diskette magazine unit)
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD  Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346 task control instruction coding description L-194 internals I-24 overview L-42 return codes L-199	M move line, \$FSEDIT line command U-233 macro assembler internal overview \$S1ASM I-253 overview S-9 macro library S-6 macro library/host S-5 magazine diskette (see 4966 diskette magazine unit) magnetic tape (see tape)
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD  Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346 task control instruction coding description L-194 internals I-24 overview L-42 return codes L-199 used with automatic initialization S-129	M move line, \$FSEDIT line command U-233 macro assembler internal overview \$S1ASM I-253 overview S-9 macro library S-6 macro library/host S-5 magazine diskette (see 4966 diskette magazine unit) magnetic tape (see tape) MASK display insert mask, \$FSEDIT
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD  Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346 task control instruction coding description L-194 internals I-24 overview L-42 return codes L-199 used with automatic initialization S-129 used with overlays S-40	M move line, \$FSEDIT line command U-233 macro assembler internal overview \$\$1ASM I-253 overview S-9 macro library S-6 macro library/host S-5 magazine diskette (see 4966 diskette magazine unit) magnetic tape (see tape) MASK display insert mask, \$FSEDIT line command U-232
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD  Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346 task control instruction coding description L-194 internals I-24 overview L-42 return codes L-199 used with automatic initialization S-129 used with overlays S-40 load mode S-149	M move line, \$FSEDIT line command U-233 macro assembler internal overview \$\$1ASM I-253 overview S-9 macro library S-6 macro library/host S-5 magazine diskette (see 4966 diskette magazine unit) magnetic tape (see tape) MASK display insert mask, \$FSEDIT line command U-232 master control block (see MCB)
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD  Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346 task control instruction coding description L-194 internals I-24 overview L-42 return codes L-199 used with automatic initialization S-129 used with overlays S-40 load mode S-149 load point defined L-40	M move line, \$FSEDIT line command U-233 macro assembler    internal overview \$S1ASM    I-253    overview S-9 macro library S-6 macro library/host S-5 magazine diskette (see 4966    diskette magazine unit) magnetic tape (see tape) MASK display insert mask, \$FSEDIT line command U-232 master control block (see MCB) Mathematical and Functional Sub-
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD  Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346 task control instruction coding description L-194 internals I-24 overview L-42 return codes L-199 used with automatic initialization S-129 used with overlays S-40 load mode S-149 load point defined L-40 load program	M move line, \$FSEDIT line command U-233 macro assembler    internal overview \$S1ASM    I-253    overview S-9 macro library S-6 macro library/host S-5 magazine diskette (see 4966    diskette magazine unit) magnetic tape (see tape) MASK display insert mask, \$FSEDIT line command U-232 master control block (see MCB) Mathematical and Functional Subroutine Library S-6
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD  Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346 task control instruction coding description L-194 internals I-24 overview L-42 return codes L-199 used with automatic initialization S-129 used with overlays S-40 load mode S-149 load point defined L-40	M move line, \$FSEDIT line command U-233 macro assembler    internal overview \$S1ASM    I-253    overview S-9 macro library S-6 macro library/host S-5 magazine diskette (see 4966    diskette magazine unit) magnetic tape (see tape) MASK display insert mask, \$FSEDIT line command U-232 master control block (see MCB) Mathematical and Functional Sub-
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD  Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346 task control instruction coding description L-194 internals I-24 overview L-42 return codes L-199 used with automatic initialization S-129 used with overlays S-40 load mode S-149 load point defined L-40 load program \$L operator command I-23, U-17	M move line, \$FSEDIT line command U-233 macro assembler     internal overview \$S1ASM     I-253     overview S-9 macro library S-6 macro library/host S-5 magazine diskette (see 4966     diskette magazine unit) magnetic tape (see tape) MASK display insert mask, \$FSEDIT     line command U-232 master control block (see MCB) Mathematical and Functional Subroutine Library S-6 MCB master control block     \$PDS S-260
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD  Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346 task control instruction coding description L-194 internals I-24 overview L-42 return codes L-199 used with automatic initialization S-129 used with overlays S-40 load mode S-149 load point defined L-40 load program \$L operator command I-23, U-17 automatic initialization	M move line, \$FSEDIT line command U-233 macro assembler     internal overview \$S1ASM     I-253     overview S-9 macro library S-6 macro library/host S-5 magazine diskette (see 4966     diskette magazine unit) magnetic tape (see tape) MASK display insert mask, \$FSEDIT     line command U-232 master control block (see MCB) Mathematical and Functional Sub- routine Library S-6 MCB master control block     \$PDS S-260     definition A-20
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD  Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346 task control instruction coding description L-194 internals I-24 overview L-42 return codes L-199 used with automatic initialization S-129 used with overlays S-40 load mode S-149 load point defined L-40 load program \$L operator command I-23, U-17 automatic initialization S-129	M move line, \$FSEDIT line command U-233 macro assembler     internal overview \$\$1ASM     I-253     overview \$-9 macro library \$-6 macro library/host \$-5 magazine diskette (see 4966     diskette magazine unit) magnetic tape (see tape) MASK display insert mask, \$FSEDIT line command U-232 master control block (see MCB) Mathematical and Functional Sub- routine Library \$-6 MCB master control block \$PDS \$-260     definition A-20     description A-28
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD  Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346 task control instruction coding description L-194 internals I-24 overview L-42 return codes L-199 used with automatic initialization S-129 used with overlays S-40 load mode S-149 load point defined L-40 load program \$L operator command I-23, U-17 automatic initialization S-129 EXEC \$JOBUTIL command U-277	M move line, \$FSEDIT line command U-233 macro assembler     internal overview \$\$1ASM     I-253     overview \$-9 macro library \$-6 macro library/host \$-5 magazine diskette (see 4966     diskette magazine unit) magnetic tape (see tape) MASK display insert mask, \$FSEDIT line command U-232 master control block (see MCB) Mathematical and Functional Sub- routine Library \$-6 MCB master control block     \$PDS \$-260     definition A-20     description A-28 MD move data base, \$DIUTIL
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD  Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346 task control instruction coding description L-194 internals I-24 overview L-42 return codes L-199 used with automatic initialization S-129 used with overlays S-40 load mode S-149 load point defined L-40 load program \$L operator command I-23, U-17 automatic initialization S-129 EXEC \$JOBUTIL command U-277 loading overlays I-22	M move line, \$FSEDIT line command U-233 macro assembler     internal overview \$\$1ASM     I-253     overview \$-9 macro library \$-6 macro library/host \$-5 magazine diskette (see 4966     diskette magazine unit) magnetic tape (see tape) MASK display insert mask, \$FSEDIT line command U-232 master control block (see MCB) Mathematical and Functional Subroutine Library \$-6 MCB master control block     \$PDS \$-260     definition A-20     description A-28 MD move data base, \$DIUTIL command U-160
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD  Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346 task control instruction coding description L-194 internals I-24 overview L-42 return codes L-199 used with automatic initialization S-129 used with overlays S-40 load mode S-149 load point defined L-40 load program \$L operator command I-23, U-17 automatic initialization S-129 EXEC \$JOBUTIL command U-277 loading overlays I-22 loading programs I-19	M move line, \$FSEDIT line command U-233 macro assembler     internal overview \$S1ASM     I-253     overview S-9 macro library S-6 macro library/host S-5 magazine diskette (see 4966     diskette magazine unit) magnetic tape (see tape) MASK display insert mask, \$FSEDIT     line command U-232 master control block (see MCB) Mathematical and Functional Subroutine Library S-6 MCB master control block     \$PDS S-260     definition A-20     description A-28 MD move data base, \$DIUTIL     command U-160 member area S-250
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD  Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346 task control instruction coding description L-194 internals I-24 overview L-42 return codes L-199 used with automatic initialization S-129 used with overlays S-40 load mode S-149 load point defined L-40 load program \$L operator command I-23, U-17 automatic initialization S-129 EXEC \$JOBUTIL command U-277 loading overlays I-22 loading programs I-19 locate data sets and overlay	M move line, \$FSEDIT line command U-233 macro assembler     internal overview \$S1ASM     I-253     overview S-9 macro library S-6 macro library/host S-5 magazine diskette (see 4966     diskette magazine unit) magnetic tape (see tape) MASK display insert mask, \$FSEDIT line command U-232 master control block (see MCB) Mathematical and Functional Subroutine Library S-6 MCB master control block     \$PDS S-260     definition A-28 MD move data base, \$DIUTIL command U-160 member area S-250 member control block (MCB) S-260
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD  Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346 task control instruction coding description L-194 internals I-24 overview L-42 return codes L-199 used with automatic initialization S-129 used with overlays S-40 load mode S-149 load point defined L-40 load program \$L operator command I-23, U-17 automatic initialization S-129 EXEC \$JOBUTIL command U-277 loading overlays I-22 loading programs I-19 locate data sets and overlay programs, \$PREFIND U-302	M move line, \$FSEDIT line command U-233 macro assembler     internal overview \$S1ASM     I-253     overview S-9 macro library S-6 macro library/host S-5 magazine diskette (see 4966     diskette magazine unit) magnetic tape (see tape) MASK display insert mask, \$FSEDIT line command U-232 master control block (see MCB) Mathematical and Functional Subroutine Library S-6 MCB master control block     \$PDS S-260     definition A-20     description A-28 MD move data base, \$DIUTIL command U-160 member area S-250 member control block (MCB) S-260 MENU
U-138 LO load indexed file, \$IAMUT1 command U-241 LOAD  Indexed Access Method CALL coding description L-344 connect file S-159 overview L-27, S-146 return codes L-346 task control instruction coding description L-194 internals I-24 overview L-42 return codes L-199 used with automatic initialization S-129 used with overlays S-40 load mode S-149 load point defined L-40 load program \$L operator command I-23, U-17 automatic initialization S-129 EXEC \$JOBUTIL command U-277 loading overlays I-22 loading programs I-19 locate data sets and overlay	M move line, \$FSEDIT line command U-233 macro assembler     internal overview \$S1ASM     I-253     overview S-9 macro library S-6 macro library/host S-5 magazine diskette (see 4966     diskette magazine unit) magnetic tape (see tape) MASK display insert mask, \$FSEDIT line command U-232 master control block (see MCB) Mathematical and Functional Subroutine Library S-6 MCB master control block     \$PDS S-260     definition A-28 MD move data base, \$DIUTIL command U-160 member area S-250 member control block (MCB) S-260

L-377	DOWN \$EDITI/N editor
internals M-9	subcommand U-189
overview C-116, L-29	TOP \$EDITI/N editor
return to primary option,	subcommand U-200
\$F\$EDIT U-223	UP \$EDIT1/N editor
menu-driven U-2	subcommand U-201 tape U-324
menus (see option selection menu)	text
(see parameter selection	\$EDIT1/N editor subcom-
menu)	mand U-195
(see primary menu)	\$FSEDIT line command
(see primary option menu)	U-233
(see secondary option menu)	volumes on disk or diskette,
(see session manager, menus)	\$MOVEVOL utility U-294
(see transaction selection	MOVE data manipulation
menu)	instruction
MENUSCRN, Multiple Terminal Manag-	coding description L-201
er C-126	overview L-19
MERGE merge data, \$FSEDIT option	MOVEA data manipulation
U-217	instruction
message, PRINTEXT instruction	coding description L-204
L-217	overview L-19
message sending utility, \$TERMUT3	MOVEBYTE subroutine, \$EDXASM
U-344	I-236
messages U-421 error U-427	MP COMP cubermand
\$DUMP U-431	move beam, \$DICOMP subcommand U-121
\$LOG U-432	move position (\$PDS) S-253
\$RMU U-433	MT move tape, \$TAPEUT1 command
\$TRAP U-435	U-324
program check U-427	MTMSTORE file, Multiple Terminal
system program check	Manager C-120, C-171, M-12
U-429	MTMSTR, Multiple Terminal Manager
IPL (see IPL messages)	C-169, C-170, M-12
Multiple Terminal Manager	multiple field format L-155
C-178	multiple program execution I-36
Remote Management Utility	multiple program structure S-26
C-279	multiple-task programs I-33
minimum execution system config-	Multiple Terminal Manager
uration S-22	accessing the terminal envi-
minimum program preparation	ronment block C-139, M-22
requirements S-22	application program C-116
mirror image description C-7, S-109	application program languages L-30
in TERMINAL configuration	application program manager
statement S-101	C-119, M-4
mixed precision combinations L-20	automatic OPEN/CLOSE C-140,
MM move block, \$FSEDIT line	M-8
command U-233	CALL
modified data S-307	ACTION C-130, L-360
modify character image tables	BEEP C-137, L-361
U <del></del> 339	CDATA C-139, L-362
modify character string, CHANGE	CHGPAN C-135, L-364
\$EDIT1/N editor subcommand	CYCLE C-132, L-365
U-184	FAN C-139, L-366
\$FSEDIT primary command U-219	FILEIO C-141, L-367
modify default storage allocation,	FTAB C-138, L-372
\$DISKUT2 U-149	LINK C-131, L-374 LINKON C-132, L-376
modifying an existing data set, \$FSEDIT U-215	MENU C-132, L-376
modifying TERMINAL statement for	SETCUR C-137, L-378
new I/O terminal I-119	SETPAN C-134, L-379
module descriptions	WRITE C-133, L-381
\$\$1ASM I-269	coding instructions L-359
supervisor I-75	components C-123, M-4
module names and entry points,	considerations for 3101
supervisor	terminal C-122
Version 1.1 S-347	data files C-120
Version 2 S-357	MTMSTORE file C-120,
move	C-171, M-12
block, MM \$FSEDIT line com-	PRGRMS volume C-120,
mand U-233	C-173
line pointer	SCRNS volume C-120, C-173
BOTTOM \$EDIT1/N editor	TERMINAL volume C-120,

direct file request types	terminal/screen management
C-144, L-370	C-117
disk file support C-140	terminal server C-119, M-7
distribution and installation	terminal support C-114, C-126
C-161	transaction oriented
dynamic screen modification	applications C-121
and creation C-136	user application programs
file control area C-142	C-124
file I/O considerations (Event	utilities C-159
Driven Executive) C-146	DISCONNECT turn off
file management C-118, M-8	specified terminals
FILEIO, disk file support	C-159
C-140	programs report C-159
FILEIO Indexed Access Method	RECONNECT turn on
considerations C-148	specified terminals
fixed screen formats C-125	C-159
functions (callable routines)	screens report C-160
C-117, C-124	terminal activity report
indexed file request types	C-159
C-144, L-369	work areas, control blocks and
indexed file support C-140,	tables M-11
L-367	buffer areas M-15, M-29
initialization programs	common area M-11, M-25
C-119, C-158, M-4, M-6	file table M-15, M-27
Input Buffer C-116, C-127	MTMSTORE data set M-12
Input Buffer Address C-116	program table M-14, M-21
Input Buffer during	screen table M-14, M-21
4978/4979/3101 buffer oper-	terminal environment block
ation C-127	(TEB) M-13, M-22
	terminal table M-13, M-21
interrupt information byte	
C-128	MULTIPLY data manipulation
messages C-178	instruction
module list M-4	coding description L-205
operation C-115	overview L-19
Output Buffer C-116	precision table L-206
Output Buffer Address C-127	multiprogramming
Output Buffer during	automatic application initial-
4978/4979/3101 buffer oper-	ization S-129
ation C-128	design feature S-13
cverview L-29, S-10	design feature S-13 multitasking I-42
overview L-29, S-10 program management C-115, M-4	
cverview L-29, S-10	
overview L-29, S-10 program management C-115, M-4	multitasking I-42
overview L-29, S-10 program management C-115, M-4 program preparation	
overview L-29, S-10 program management C-115, M-4 program preparation COBOL C-166	multitasking I-42
overview L-29, S-10 program management C-115, M-4 program preparation COBOL C-166 Event Driven Language	multitasking I-42
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165	multitasking I-42
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165 PL/I C-167	multitasking I-42  N NE (not equal) L-34
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations	NE (not equal) L-34 newline subroutine, terminal I/0
overview L-29, S-10 program management C-115, M-4 program preparation    C0BOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    C0BOL C-153	NE (not equal) L-34 newline subroutine, terminal I/0 I-112
overview L-29, S-10 program management C-115, M-4 program preparation     COBOL C-166     Event Driven Language     C-164     FORTRAN C-165     PL/I C-167 programming considerations     COBOL C-153     Event Driven Language	NE (not equal) L-34 newline subroutine, terminal I/0 I-112 NEXTQ queue processing
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    COBOL C-153    Event Driven Language    C-151	NE (not equal) L-34 newline subroutine, terminal I/0 I-112 NEXTQ queue processing instruction
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    COBOL C-153    Event Driven Language    C-151    FORTRAN C-152	NE (not equal) L-34 newline subroutine, terminal I/0 I-112 NEXTQ queue processing instruction coding description L-207
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    COBOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155	NE (not equal) L-34 newline subroutine, terminal I/0 I-112 NEXTQ queue processing instruction coding description L-207 overview L-37, S-32
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    COBOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155 return codes (FILEIO) C-145,	NE (not equal) L-34 newline subroutine, terminal I/0 I-112 NEXTQ queue processing instruction coding description L-207 overview L-37, S-32 NOFLOAT floating-point command
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    COBOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155 return codes (FILEIO) C-145, L-371	NE (not equal) L-34 newline subroutine, terminal I/0 I-112 NEXTQ queue processing instruction coding description L-207 overview L-37, S-32 NOFLOAT floating-point command entries module description I-79
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    COBOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155 return codes (FILEIO) C-145, L-371 screen definition C-121	NE (not equal) L-34 newline subroutine, terminal I/0 I-112 NEXTQ queue processing instruction coding description L-207 overview L-37, S-32 NOFLOAT floating-point command entries module description, \$EDXASM
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    COBOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155 return codes (FILEIO) C-145, L-371 screen definition C-121 screen formats C-125	NE (not equal) L-34 newline subroutine, terminal I/O I-112 NEXTQ queue processing instruction coding description L-207 overview L-37, S-32 NOFLOAT floating-point command entries module description I-79 NOLIST no list option, \$EDXASM command U-358
overview L-29, S-10 program management C-115, M-4 program preparation    C0BOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    C0BOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155 return codes (FILEIO) C-145, L-371 screen definition C-121 screen formats C-125    IPLSCRN C-125	NE (not equal) L-34 newline subroutine, terminal I/0 I-112 NEXTQ queue processing instruction coding description L-207 overview L-37, S-32 NOFLOAT floating-point command entries module description I-79 NOLIST no list option, \$EDXASM command U-358 NOMSG message suppression,
overview L-29, S-10 program management C-115, M-4 program preparation    C0BOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    C0BOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155 return codes (FILEIO) C-145, L-371 screen definition C-121 screen formats C-125    IPLSCRN C-125    MENUSCRN C-126	NE (not equal) L-34 newline subroutine, terminal I/O I-112 NEXTQ queue processing instruction coding description L-207 overview L-37, S-32 NOFLOAT floating-point command entries module description I-79 NOLIST no list option, \$EDXASM command U-358 NOMSG message suppression, \$JOBUTIL command U-282
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    COBOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155 return codes (FILEIO) C-145, L-371 screen definition C-121 screen formats C-125    IPLSCRN C-125    MENUSCRN C-126    SCRNSREP C-126	NE (not equal) L-34 newline subroutine, terminal I/O I-112 NEXTQ queue processing instruction coding description L-207 overview L-37, S-32 NOFLOAT floating-point command entries module description I-79 NOLIST no list option, \$EDXASM command U-358 NOMSG message suppression, \$JOBUTIL command U-282 non-compressed byte string S-309
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    COBOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155 return codes (FILEIO) C-145, L-371 screen definition C-121 screen formats C-125    IPLSCRN C-125    MENUSCRN C-126    SCRNSREP C-126    SIGNONSC C-126	NE (not equal) L-34 newline subroutine, terminal I/O I-112 NEXTQ queue processing instruction    coding description L-207    overview L-37, S-32 NOFLOAT floating-point command entries module description I-79 NOLIST no list option, \$EDXASM command U-358 NOMSG message suppression, \$JOBUTIL command U-282 non-compressed byte string S-309 non-labeled tapes
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    COBOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155 return codes (FILEIO) C-145, L-371 screen definition C-121 screen formats C-125    IPLSCRN C-125    MENUSCRN C-126    SCRNSREP C-126    SIGNONSC C-126 screen panel manager M-7	NE (not equal) L-34 newline subroutine, terminal I/O I-112 NEXTQ queue processing instruction coding description L-207 overview L-37, S-32 NOFLOAT floating-point command entries module description I-79 NOLIST no list option, \$EDXASM command U-358 NOMSG message suppression, \$JOBUTIL command U-282 non-compressed byte string S-309 non-labeled tapes description S-241
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    COBOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155 return codes (FILEIO) C-145, L-371 screen definition C-121 screen formats C-125    IPLSCRN C-125    MENUSCRN C-126    SCRNSREP C-126    SIGNONSC C-126 screen panel manager M-7 SIGNON/SIGNOFF C-156	NE (not equal) L-34 newline subroutine, terminal I/O I-112 NEXTQ queue processing instruction coding description L-207 overview L-37, S-32 NOFLOAT floating-point command entries module description I-79 NOLIST no list option, \$EDXASM command U-358 NOMSG message suppression, \$JOBUTIL command U-282 non-compressed byte string S-309 non-labeled tapes description S-241 layout S-242
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    COBOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155 return codes (FILEIO) C-145, L-371 screen definition C-121 screen formats C-125    IPLSCRN C-125    MENUSCRN C-126    SCRNSREP C-126    SIGNONSC C-126 screen panel manager M-7 SIGNON/SIGNOFF C-156 SIGNONFL C-174	NE (not equal) L-34 newline subroutine, terminal I/O I-112 NEXTQ queue processing instruction coding description L-207 overview L-37, S-32 NOFLOAT floating-point command entries module description I-79 NOLIST no list option, \$EDXASM command U-358 NOMSG message suppression, \$JOBUTIL command U-282 non-compressed byte string S-309 non-labeled tapes description S-241 layout S-242 processing S-243
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    COBOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155 return codes (FILEIO) C-145, L-371 screen definition C-121 screen formats C-125    IPLSCRN C-125    MENUSCRN C-126    SCRNSREP C-126    SIGNONSC C-126 screen panel manager M-7 SIGNON/SIGNOFF C-156    SIGNONFL C-174 storage requirements C-168	NE (not equal) L-34 newline subroutine, terminal I/O I-112 NEXTQ queue processing instruction coding description L-207 overview L-37, S-32 NOFLOAT floating-point command entries module description I-79 NOLIST no list option, \$EDXASM command U-358 NOMSG message suppression, \$JOBUTIL command U-282 non-compressed byte string S-309 non-labeled tapes description S-241 layout S-242 processing S-243 NOTE disk/tape I/O instruction
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    COBOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155 return codes (FILEIO) C-145, L-371 screen definition C-121 screen formats C-125    IPLSCRN C-126    SCRNSREP C-126    SCRNSREP C-126    SIGNONSC C-126 screen panel manager M-7 SIGNON/SIGNOFF C-156    SIGNONFL C-174 storage requirements C-168 swap out data set C-116	NE (not equal) L-34 newline subroutine, terminal I/O I-112 NEXTQ queue processing instruction coding description L-207 overview L-37, S-32 NOFLOAT floating-point command entries module description I-79 NOLIST no list option, \$EDXASM command U-358 NOMSG message suppression, \$JOBUTIL command U-282 non-compressed byte string S-309 non-labeled tapes description S-241 layout S-242 processing S-243 NOTE disk/tape I/O instruction coding description L-209
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    COBOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155 return codes (FILEIO) C-145, L-371 screen definition C-121 screen formats C-125    IPLSCRN C-125    MENUSCRN C-126    SCRNSREP C-126    SCRNSREP C-126    SIGNONSC C-126 screen panel manager M-7 SIGNON/SIGNOFF C-156    SIGNONFL C-174 storage requirements C-168 swap out data set C-116 system generation	NE (not equal) L-34 newline subroutine, terminal I/O I-112 NEXTQ queue processing instruction coding description L-207 overview L-37, S-32 NOFLOAT floating-point command entries module description I-79 NOLIST no list option, \$EDXASM command U-358 NOMSG message suppression, \$JOBUTIL command U-282 non-compressed byte string S-309 non-labeled tapes description S-241 layout S-242 processing S-243 NOTE disk/tape I/O instruction coding description L-209 overview L-22
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    COBOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155 return codes (FILEIO) C-145, L-371 screen definition C-121 screen formats C-125    IPLSCRN C-125    MENUSCRN C-126    SCRNSREP C-126    SIGNONSC C-126 screen panel manager M-7 SIGNON/SIGNOFF C-156    SIGNONFL C-174 storage requirements C-168 swap out data set C-116 system generation considerations C-169	NE (not equal) L-34 newline subroutine, terminal I/O I-112 NEXTQ queue processing instruction    coding description L-207    overview L-37, S-32 NOFLOAT floating-point command entries module description I-79 NOLIST no list option, \$EDXASM    command U-358 NOMSG message suppression, \$JOBUTIL command U-282 non-compressed byte string S-309 non-labeled tapes    description S-241    layout S-242    processing S-243 NOTE disk/tape I/O instruction    coding description L-209    overview L-22 notify of an event (see POST)
overview L-29, S-10 program management C-115, M-4 program preparation    C0BOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    C0BOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155 return codes (FILEIO) C-145, L-371 screen definition C-121 screen formats C-125    IPLSCRN C-125    MENUSCRN C-126    SCRNSREP C-126    SIGNONSC C-126 screen panel manager M-7 SIGNON/SIGNOFF C-156    SIGNONFL C-174 storage requirements C-168 swap out data set C-116 system generation considerations C-169    data set requirements	NE (not equal) L-34 newline subroutine, terminal I/O I-112 NEXTQ queue processing instruction    coding description L-207    overview L-37, S-32 NOFLOAT floating-point command entries module description I-79 NOLIST no list option, \$EDXASM    command U-358 NOMSG message suppression, \$JOBUTIL command U-282 non-compressed byte string S-309 non-labeled tapes    description S-241    layout S-242    processing S-243 NOTE disk/tape I/O instruction    coding description L-209    overview L-22 notify of an event (see POST) NQ reset prompt mode, \$COPYUT1
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    COBOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155 return codes (FILEIO) C-145, L-371 screen definition C-121 screen formats C-125    IPLSCRN C-125    MENUSCRN C-126    SCRNSREP C-126    SCRNSREP C-126    SIGNONSC C-126 screen panel manager M-7 SIGNON/SIGNOFF C-156    SIGNONFL C-174 storage requirements C-168 swap out data set C-116 system generation considerations C-169    data set requirements    C-171, C-175	NE (not equal) L-34 newline subroutine, terminal I/O I-112 NEXTQ queue processing instruction coding description L-207 overview L-37, S-32 NOFLOAT floating-point command entries module description I-79 NOLIST no list option, \$EDXASM command U-358 NOMSG message suppression, \$JOBUTIL command U-282 non-compressed byte string S-309 non-labeled tapes description S-241 layout S-242 processing S-243 NOTE disk/tape I/O instruction coding description L-209 overview L-22 notify of an event (see POST) NQ reset prompt mode, \$COPYUT1 command U-64
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    COBOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155 return codes (FILEIO) C-145, L-371 screen definition C-121 screen formats C-125    IPLSCRN C-125    MENUSCRN C-126    SCRNSREP C-126    SCRNSREP C-126    SIGNONSC C-126 screen panel manager M-7 SIGNON/SIGNOFF C-156    SIGNONFL C-174 storage requirements C-168 swap out data set C-116 system generations C-169    data set requirements    C-171, C-175    volume requirements C-169	NE (not equal) L-34 newline subroutine, terminal I/O I-112 NEXTQ queue processing instruction coding description L-207 overview L-37, S-32 NOFLOAT floating-point command entries module description I-79 NOLIST no list option, \$EDXASM command U-358 NOMSG message suppression, \$JOBUTIL command U-282 non-compressed byte string S-309 non-labeled tapes description S-241 layout S-242 processing S-243 NOTE disk/tape I/O instruction coding description L-209 overview L-22 notify of an event (see POST) NQ reset prompt mode, \$COPYUT1 command U-64 nucleus initialization I-15
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    COBOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155 return codes (FILEIO) C-145, L-371 screen definition C-121 screen formats C-125    IPLSCRN C-126    SCRNSREP C-126    SCRNSREP C-126    SIGNONSC C-126 screen panel manager M-7 SIGNON/SIGNOFF C-156    SIGNONFL C-174 storage requirements C-168 swap out data set C-116 system generation    considerations C-169    data set requirements    C-171, C-175    volume requirements C-169 terminal environment block	NE (not equal) L-34 newline subroutine, terminal I/O I-112 NEXTQ queue processing instruction coding description L-207 overview L-37, S-32 NOFLOAT floating-point command entries module description I-79 NOLIST no list option, \$EDXASM command U-358 NOMSG message suppression, \$JOBUTIL command U-282 non-compressed byte string S-309 non-labeled tapes description S-241 layout S-242 processing S-243 NOTE disk/tape I/O instruction coding description L-209 overview L-22 notify of an event (see POST) NQ reset prompt mode, \$COPYUT1 command U-64 nucleus initialization I-15 null character U-253
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    COBOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155 return codes (FILEIO) C-145, L-371 screen definition C-121 screen formats C-125    IPLSCRN C-126    SCRNSREP C-126    SIGNONSC C-126    SIGNONSC C-126 screen panel manager M-7 SIGNON/SIGNOFF C-156    SIGNONFL C-174 storage requirements C-168 swap out data set C-116 system generation    considerations C-169    data set requirements    C-171, C-175    volume requirements C-169 terminal environment block (TEB) C-128, M-13	NE (not equal) L-34 newline subroutine, terminal I/O I-112 NEXTQ queue processing instruction coding description L-207 overview L-37, S-32 NOFLOAT floating-point command entries module description I-79 NOLIST no list option, \$EDXASM command U-358 NOMSG message suppression, \$JOBUTIL command U-282 non-compressed byte string S-309 non-labeled tapes description S-241 layout S-242 processing S-243 NOTE disk/tape I/O instruction coding description L-209 overview L-22 notify of an event (see POST) NQ reset prompt mode, \$COPYUT1 command U-64 nucleus initialization I-15
overview L-29, S-10 program management C-115, M-4 program preparation    COBOL C-166    Event Driven Language    C-164    FORTRAN C-165    PL/I C-167 programming considerations    COBOL C-153    Event Driven Language    C-151    FORTRAN C-152    PL/I C-155 return codes (FILEIO) C-145, L-371 screen definition C-121 screen formats C-125    IPLSCRN C-126    SCRNSREP C-126    SCRNSREP C-126    SIGNONSC C-126 screen panel manager M-7 SIGNON/SIGNOFF C-156    SIGNONFL C-174 storage requirements C-168 swap out data set C-116 system generation    considerations C-169    data set requirements    C-171, C-175    volume requirements C-169 terminal environment block	NE (not equal) L-34 newline subroutine, terminal I/O I-112 NEXTQ queue processing instruction coding description L-207 overview L-37, S-32 NOFLOAT floating-point command entries module description I-79 NOLIST no list option, \$EDXASM command U-358 NOMSG message suppression, \$JOBUTIL command U-282 non-compressed byte string S-309 non-labeled tapes description S-241 layout S-242 processing S-243 NOTE disk/tape I/O instruction coding description L-209 overview L-22 notify of an event (see POST) NQ reset prompt mode, \$COPYUT1 command U-64 nucleus initialization I-15 null character U-253

null representation, defining U-253 number representation conversion (see conversion)

0

object data set for \$EDXASM U-357 object module record format, U-407 object text elements, format of, \$EDXASM I-215 OFF (set tape offline) L-75 OFF remove breakpoints and trace ranges, \$DEBUG command U-97 OLE operand list element, \$EDXASM format of I-216 in instruction parsing (\$EDXASM) I-220 used in \$IDEF I-241 online debug aids S-67 op (operation field) L-3 OPCHECK subroutine, \$EDXASM I-232 opcode table, instruction parsing (\$EDXASM) I-220, I-223 open a data set disk or diskette I-90 tape I-99 open EXIO device, EXOPEN I-125 open member (\$PD\$) S-261 OPENIN Host Communications Facility, TP operand C-93 **OPENOUT Host Communications** Facility, TP operand C-94 operands defined L-3 keyword L-5 parameter naming (Px) L-8 operating conventions, supervisor program I-67 operating environment S-22 operation code, instruction parsing (\$EDXASM) I-220 operation codes, Event Driven Language I-68 operations using index registers L-20 operator commands S-63, U-9 operator signals, terminal I/O option selection menu U-33 optional features support L-15 OTE define object text element \$EDXASM instruction I-227 OUTPUT \$LINK control record U-399 Output Buffer, Multiple Terminal Manager C-116, C-128 contents during 4978/4979/3101 buffer operation C-129 definition M-29 overflow L-20 overlay function processor table I-167, I-220 overlay program S-40 instructions, \$EDXASM I-259 loading I-22 locating, \$PREFIND U-302 subroutines, \$EDXASM I-231 user I-38 overlay program execution I-38 overlay selection, instruction

parsing (\$EDXASM) I-223

overlay table I-167, I-220 overview data definition statements L-17 data formatting instructions L-18 data format module names L-18 data manipulation instructions L-19 data representation L-19 mixed-precision operations L-20 operations using index registers L-20 overflow L-20 vector L-19 disk I/O instructions L-22 EXIO control instructions L-24 floating-point arithmetic L-20 floating-point arithmetic instructions L-20 data representation L-21 operations using index registers L-21 return codes L-21 graphics instructions L-26 Indexed Access Method instructions L-27 instructions and statements L-15 integer and logical instructions L-19 listing control statements L-28 Multiple Terminal Manager instructions L-29 program control statements L-32 program module sectioning statements L-33 program sequencing instructions L-34 queue processing L-37 sensor-based I/O statements L-39 single-precision L-19 system configuration statements L-39 tape I/O instructions L-40 task control instructions L-42 terminal I/O instructions L-44 timing instructions L-50

Р

P/I (see process interrupt)
PA patch, \$DISKUT2 command U-147
page eject S-63, U-16
parameter equate tables L-11
parameter naming operands in the instruction format L-8
parameter passing, Remote
Management Utility C-212
parameter selection menu U-33
parameter tables, control block and L-11

PARM program parameter passing,	U-100
\$JOBUTIL command U-283 parsing, input data (\$EDXASM)	task control instruction coding description L-213
I-218	internals I-58
partition assignment changing, \$CP	overview L-42, S-34
operator command U-14	supervisor function I-46
partitioned data sets \$-247	power outage, restoring after
partitions S-42	S-129
PASSTHRU function	PR print member, \$DICOMP command
conducting a session C-227	U-108
establishing a session C-225 internals I-166, I-179	precision L-19 floating-point arithmetic
overview C-225	L-21
programming considerations	integer and logical L-19
C-237	precision combinations,
sample program C-265	allowed L-20
types of records C-232	precision table
virtual terminals C-239 Passthru record C-209	ADD L-53 ADDV L-54
patch	DIVIDE L-101
disk/diskette, PA \$DISKUT2	MULTIPLY L-206
command U-147	overview L-20
Remote Management Utility	SUBTRACT L-284
defaults C-283	prefind U-302
storage, \$P operator command S-63, U-18	PREPARE IDCB command L-175 PRGRMS volume, Multiple Terminal
storage or registers, PATCH	Manager C-120, C-173
\$DEBUG command U-98	primary
PATCH modify storage or registers,	commands, \$FSEDIT U-218
\$DEBUG, command U-98	option menu, \$FSEDIT U-213
PAUSE operator intervention,	option menu, session manager
\$JOBUTIL command U-284 PC plot curve	S-218, U-35 task
\$DICOMP subcommand U-119	internals I-29
from plot curve data member	overview S-29
(\$PDS) S-255	volume S-60
PD pulse DO, \$IOTEST command	primary-level index block
U-265	description S-195 overview S-151
PF,code TERMCTRL function L-288 PF keys (see program function	PRINDATE terminal I/O instruction
keys)	coding description L-215
phase execution and loading,	overview L-44, S-47
\$\$1ASM I-255	timer-related instruction
PI process interrupt (see process	\$-33 PRINT 1: -t:
interrupt) U-267 PID program directory S-27	PRINT listing control statement coding description L-216
PIXB (see primary-level index	overview L-28
block)	print member, PR \$DICOMP command
PL/I	U-108
execution requirements S-24	PRINTEXT terminal I/O instruction
link editing S-71	coding description L-217 overview L-44, S-47
overview S-8 program preparation	return codes L-219
requirements S-23	PRINTIME terminal I/O instruction
supported by Multiple Terminal	coding description L-221
Manager C-200	overview L-44, L-50, S-47
PL plot data, \$DICOMP subcommand	timer-related instruction
U-122 plot control block (see PLOTCB)	S-33 PRINTNUM terminal I/O instruction
plot curve data member (\$PDS)	coding description L-222
S-251	overview L-44, S-47
PLOTCB graphics plot control	PRINTON define terminal name,
block L-210	\$RJE2780/\$RJE3780 C-75
PLOTGIN graphics instruction	priority
coding description L-210 overview L-26	assigned to tasks S-29 design feature S-13
POINT	illustrated S-38
disk/tape instruction	internals I-31
coding description L-212	task L-226, L-286
overview L-22, S-54	PROC identify nested procedure,
point-to-point (BSC) S-65	\$JOBUTIL command U-286
point-to-point vector drawing S-46	procedures, session manager (see
POST	session manager) PROCESS Indexed Access Method
post an event, \$DEBUG command	CALL

coding description L-347	program structure S-36
overview L-27, S-147	internals I-33
return codes L-349	program/task concepts I-29, S-29
process interrupt	PROGRAM task control instruction
control block (SBIOCB) I-128	coding description L-225
description S-48	internals I-30
IODEF statement L-189	overview L-42, S-31
IOTEST command U-267	PROGSTOP task control statement
supported by sensor I/O S-15	coding description L-234
user routine (SPECPI) L-189	overview L-42, S-31
process mode	prompting and advance input,
definition 5-150	terminal I/O L-46
processing compiler output with	protected field S-307, U-253
\$LINK or \$UPDATE U-360	protocol, BSC transmission I-156
processor status word (see PSW)	PSW processor status word U-430
PROGEQU L-13	PU PUNCHO/PUNCHS function, \$RJE2780/\$RJE3780 reset type
program equates I-312	C-76
assembly/compilation U-352	pulse a digital output address, PD
control L-32	\$IOTEST command U-264
disabling S-102	PUNCHO/PUNCHS define output file,
entry (see \$FSEDIT, \$EDIT1/N)	\$RJE2780/\$RJE3780 C-75
function (PF) keys L-47	purpose of EDL L-1
internals I-108	PUT Indexed Access Method CALL
listing, KEYS \$IMAGE	coding description L-350
command U-253	overview L-27
listing 4978, \$PFMAP	return codes L-351
utility U-301	PUTDE Indexed Access Method CALL
when using \$FONT edit	coding description L-352
mode U-206	overview L-27
when using \$FSEDIT U-211	return codes L-353
when using \$IMAGE edit	PUTEDIT data formatting
mode U-255	instruction
when using session	coding description L-236
manager U-28	overview L-18
header I-30	return codes L-238
identifier, \$JOBUTIL command	PUTSTORE TERMCTRL function L-288
U-287	PUTUP Indexed Access Method CALL
internal processing I-30	coding description L-354
library update (see \$UPDATE)	overview L-27
load process, \$PREFIND U-302	return codes L-355
loading (see also LOAD) I-19	Px L-8
module sectioning functions	
L-33 organization S-29	Transfer to the second
sequencing functions L-34	Q
structure S-29	
termination, EXIO I-126	
types I-32	QCB task control statement 5-33
program check error messages	coding description L-240
U-427	overview L-42
program execution via Remote Man-	queue control block I-45,
agement Utility	I-54
EXEC function C-220	QD queue descriptor I-64, L-37
PASSTHRU function C-225	QE queue entry
SHUTDOWN function C-251	functions I-64
PROGRAM identifier, \$JOBUTIL	overview L-37
command U-287	processing S-32
program preparation	QUALIFY modify base address,
\$EDXASM I-211, U-356	\$DEBUG command U-101
\$\$1ASM I-253, U-372	QUESTION terminal I/O instruction
host assembler U-382	coding description_L-242
of Remote Management Utility	overview L-44, S-47
I-184	queuable resource 5-33
summary S-18	queue control block (see QCB)
usage example \$-367	queue descriptor (see QD)
Program Preparation Facility	queue entry (see QE)
description S-71	queue processing I-64
overview S-5	queue processing instructions
program preparation utilities	L-37
U-351	queue processing support module,
program preparation utilities (session manager) OU-36, S-214	QUEUEIO, description I-81
	QUEUEIO queue processing support module description I-81
program/storage manager, Multiple Terminal Manager M-4	modute description 1-01
reiminat nanager 11 T	

R	realtime data member \$PDS S-251
LK	RT \$DICOMP subcommand U-124
	RECEIVE function
RA reassign address, \$TERMUT1 com-	overview C-243
mand U-336	sample program C-262
random access S-53 random work file operation,	RECONNECT Multiple Terminal Manager utility C-120, C-159
\$\$1ASM I-260	record
RCB (see Remote Management	blocking, Remote Management
Utility, control block)	Utility C-211
RDCURSOR terminal I/O instruction	definition_ S-53
coding description L-244	exchange, Remote Management
overview L-44, S-47 RE	Utility C-208 format for object module,
copy from basic exchange data	\$LINK U-407
set, \$COPY command U-59	header, Remote Management
rename, \$TERMUT1 command	Utility C-209
/U-337 rename member, \$DISKUT1 com-	sizes, Host Communications Facility C-83
mand U-135, U-136	reformat diskettes U-68
rename member, \$DIUTIL	register, index L-6
command U-161	register, software L-6
reset parameters, \$IAMUT1	register conventions
command U-243 restore 4974 to standard	\$\$1ASM I-257 BSCAM processing I-147
character set, \$TERMUT2	common emulator setup routine
U-339	I-68
read	EBCDIC to floating-point
analog input, AI \$IOTEST	conversion I-205
U-268 character image table from	summary chart \$S1ASM I-258 terminal I/O support I-106
4978, GET \$FONT U-206	REL release a status record,
data set into work file	\$HCFUT1 command C-110
\$EDIT1 U-177	relational statements L-180
\$EDITIN U-176	RELEASE
\$FSEDIT U-216 digital input, DI \$IOTEST	Host Communications Facility, TP operand C-96
command U-266	Indexed Access Method CALL
digital input using external	5-147
U-266	coding description L-356
Host Communications Facility,	overview L-27, S-147
TP operand C-95 IDCB command L-175	return codes L-357 release a status record, REL
operations (BSC) I-157	\$HCFUT1 command C-110
program, RP command	release space (\$PDS) S-261
\$UPDATE U-410	relocating program loader I-19
\$UPDATEH U-419 READ instruction	relocation dictionary, \$EDXASM I-250
disk/diskette return codes	REMARK operator comment, \$JOBUTIL
L-249, U-455	command U-288
disk/diskette/tape I/O	remote job entry to host,
instruction coding description L-245	\$RJE2780/\$RJE3780 C-73 Remote Management Utility
overview L-22	CDRRM equates C-292
tape return codes L-249,	control block (RCB)
U-456	description I-164, I-169
READDATA read data from host,	equate tables C-292,
\$HCFUT1 command C-108 READID IDCB command L-175	I-295
READOBJ read object module,	use in problem determi- nation I-190
\$HCFUT1 command C-109	defaults C-283
READTEXT terminal I/O instruction	error handling C-277
coding description L-251	function table I-164, I-167
overview L-44, S-48 return codes L-255	functions C-206, I-166 installation C-281
return codes, virtual terminal	interface C-207
communications L-256	internals I-216
ready a task supervisor execution	logic flow I-170
state I-43	messages C-279
READ1 IDCB command L-175 READ80 read 80 byte records,	modifying defaults C-283 module descriptions I-191
\$HCFUT1 command C-109	module list I-186
real image ACCA terminals C-7	operation C-213
	avantay function processor

table I-167, I-220	FADD L-136
overlay table I-167, I-220	FDIVD L-138
overview C-205	FILEIO C-145
program preparation I-184	floating point instruction
requirements C-207	U-450
sample host programs C-259	FMULT L-145
system generation	formatted screen image U-450
considerations C-281	FSUB L-160
TERMINAL statement example	FTAB C-138, L-373
S-107	GET L-340
terminating C-251	GETSEQ L-343
remote system (see Remote	in Remote Management Utility
Management Utility) C-205	control block I-190
remove breakpoints and trace	Indexed Access Method U-451
ranges, OFF \$DEBUG command U-97	LOAD L-199, U-452
rename member	LOAD (Indexed Access Method)
RE \$DISKUT1 command U-135, U-136	L-346
RE \$DIUTIL command U-161	Multiple Terminal Manager U-453
RENUM renumber lines	PRINTEXT L-219
\$EDIT1/N subcommand U-196	PROCESS L-349
	PUT L-351
\$FSEDIT primary command U-224 reorganize an indexed data set	PUTDE L-353
U-242	PUTEDIT L-238
procedure S-166	PUTUP L-355
report data member (\$PDS) S-251	READ disk/diskette L-249,
reposition line pointer (see move	U-455
line pointer)	READ tape L-250, U-456
Request record C-209	READTEXT L-255
reserved labels L-4	RELEASE L-357
reset	SBIO U-457
function, \$RJE2780/\$RJE3780	SBIO instruction L-262
attention request C-76	SETPAN C-135
IDCB command L-176	tape L-77
Indexed Access Method	TERMCTRL L-288
ECHO mode, EC \$IAMUT1 com-	terminal I/O L-255, U-458
mand <b>U-240</b>	ACCA U-459
SE command parameters, RE	interprocessor
\$IAMUT1 command U-243	communications C-31,
line command, \$FSEDIT primary	U-460
command U-225	virtual terminal L-256,
RESET task control instruction	U-461
coding description_L-258	TP (Host Communications Facil-
overview L-42, S-31	ity) C-102, U-463 WHERES L-316
resident assembler routines I-256	
resolution, enhanced I-201	WRITE disk/diskette L-320, U-455
resolution, standard graphics I-201	WRITE tape L-320, U-456
resource control, supervisor I-54	return from immediate action
restart, automatic S-129	routine (SUPEXIT) I-49
restore	return from task level (SUPRTURN)
disk or disk volume from tape,	I-49
RT \$TAPEUT1 command U-326	RETURN program control
dump volume utility, \$MOVEVOL	instruction
U-294	coding description L-259
4974 to standard character	overview L-32, S-31
set, RE \$TERMUT2 command	supervisor entry point I-279,
U-343	I-313
resulting field (EOR) L-122	supervisor interface I-62
return codes (see also completion	REW (rewind tape) L-75
codes)	rewind tape, MT \$TAPEUT1 command
\$DISKUT3 S-319, U-444	U-324
\$PDS U-445	RH reassign hardcopy, \$TERMUT1
BSC C-57, U-446	_command U-338
CONVTB L-80	RI read
CONVTD L-83	transparent/non-transparent,
data formatting instructions	\$BSCUT2 command C-68
U-447	RJE (see Remote Job Entry)
DELETE L-330	RLOADER I-19, I-22
DISCONN L-333 ENDSEQ L-335	RLOADER/RLOADRU module
EXIO U-448	description I-78 RO reorganize indexed file,
EXIO U-448 EXIO instruction L-131	\$IAMUT1 command U-242
EXIO instruction L-131 EXIO interrupt L-132	ROFF (rewind offline) L-75
EXTRACT L-337	KOLL (LEMING OLITHIE) F=13
CATROVE E JUL	

roll screen, terminal I/O L-48, S-293 RP read program \$UPDATE command U-410 \$UPDATEH command U-419 RPQ D02038, 4978 display station attachment C-6, S-97 different device configurations RSTATUS IDCB command L-175 activate realtime data member, \$DICOMP subcommand U-124 change realtime data member S-258 name (\$PDS) disk or disk volume from tape, \$TAPEUT1 utility U-326 RWI read/write non-transparent, \$BSCUT2 command C-58 RWIV read/write non-transparent conversational, \$BSCUT2 C-71 RWIVX read/write transparent conversational, \$BSCUT2 C-70 RWIX read/write transparent, \$BSCUT2 command C-67 RWIXMP read/write multidrop transparent, \$BSCUT2 command C-60 S save data, \$DICOMP subcommand U-124

SAVE data set on disk, \$IMAGE command U-254 work data set, \$EDIT1/N subcommand U-197 save current task status (TASKSAVE) I-54 save data, SA \$DICOMP subcommand U-124 save disk or disk volume on tape, \$TAPEUT1 utility U-330 save storage and registers, \$TRAP utility U-348 SB special PI bit, \$IOTEST command U-267 SBAI sensor based I/O support I-80 module description SBAO sensor based I/O support module description I-80 SBCOM sensor based I/O support I-80 module description SBDIDO sensor based I/O support module description I-80 SBIO sensor based I/O instruction coding description L-260 control block (SBIOCB) overview L-39, S-51 return codes L-262 I-127 SBIOCB sensor based I/O control block I-127

SBPI sensor based I/O support module description I-80

SC save control store, \$TERMUT2

coding description L-270

screen format builder utility,

\$IMAGE S-68, U-250 SCREEN graphics instruction

overview L-26

screen image format building U-250 screen images, retrieving and displaying S-300 screen management, terminal I/O L-48 SCRNS volume, Multiple Terminal
Manager C-120, C-173
SCRNSREP, Multiple Terminal
Manager C-125
Manager C-125 scrolling, \$FSEDIT U-210 SCSS IDCB command L-176 SE set parameters, \$IAMUT1 command U-244 SE set status, \$HCFUT1 command C-110 second-level index block description S-197 overview S-153 secondary disk volumes S-132 volumes S-60 secondary option menus S-218, (see session manager) sectioning of program modules L-33 sector 5-52 self-defining terms L-4 send data, HX \$DICOMP subcommand U-118 data set, SEND function C-247 message to another terminal, \$TERMUT3 utility U-344 SEND function internals I-166, I-172 overview C-247 sample program sensor based I/O assignment L-188 I/O control block (SBIOCB) I-127 modules (IOLOADER/IOLOADRU) I-78 statement overview L-39 support module descriptions I-81 symbolic L-9 SENSORIO configuration statement S-51, S-84 sequence chaining L-27 sequencing instructions, program L-34 sequential access in Indexed Access Method S-145 overview S-53 sequential work file operations (\$S1ASM) I-259 serially reusable resource (SRR) I-59, S-33 session, PASSTHRU conducting C-227 establishing C-225 C-230 logic flow diagram using \$DEBUG utility C-272 session manager U-27 \$SMALLOC data set allocation S-222, U-30 control data set \$SMDELET data set deletion control data set S-222, U-32 adding an option S-209, S-224 adding an option communications utilities U-42 communications utilities

command U-343

5-217	set tape offline, MT \$TAPEUT1 com-
data management S-215	mand U-324
diagnostic utilities S-217	set time, \$T operator command U-19
disk utilities (see data	SETBUSY supervisor busy routine
management)	I-48, I-63
execute program utilities	SETCUR, Multiple Terminal Manager
S-216	CALL
graphics utilities S-216	coding description C-137,
job stream processor	L-378
utilities S-216	internals M-9
logon menu U-27	overview C-117, L-29
primary S-218, U-35	SETEOD subroutine S-324
program preparation	SETPAN, Multiple Terminal Manager
utilities S-214	CALL
secondary S-218, U-36 summary of S-213	coding description C-134, L-379
terminal utilities S-215	internals M-9
updating primary option	overview C-117, L-29
S-224	return codes L-380
creating a new menu 5-224	setup procedure for \$JOBUTIL
data management U-38	U-271
data set deletion U-32	SG special PI group, \$IOTEST com-
data sets creation U-29	mand U-267
diagnostic utilities U-43	SHIFTL data manipulation
execute program utilities	instruction
U-41	coding description L-271
graphics utilities U-40	overview L-19
invoking U-27	SHIFTR data manipulation
invoking a \$JOBUTIL procedure 5-229	instruction coding description L-273
job stream processor	overview L-19
utilities U-42	SHUTDOWN function C-251, I-166,
menus U-33	I-181
minimum partition size	SI save image store, \$TERMUT2 com-
required U-27	mand U-341
operational overview 5-209	SIGNON/SIGNOFF, Multiple Terminal
primary option menu, \$SMMPRIM	Manager C-156
S-218, U-35	SIGNONFL C-174
procedures	single program execution I-35
communications utilities	single-task program I-33
S-217	single task program 5-34 SIXB (see second-level index
data management utilities S-215	block)
diagnostic utilities	SLE sublist element, \$EDXASM
5-217	format of I-217
execute program utilities	in instruction parsing
S-216	(\$EDXASM) I-220
graphics utilities S-216	instruction description and
job stream processor	format I-229
utilities S-216	used in \$IDEF I-241
overview S-220	software register L-6
program preparation utilities S-214	software trace table S-265 sort/merge S-9
terminal utilities S-215	source program compiling S-71
updating S-225	source program entry and editing
program function keys U-28	S-66, U-351
program preparation utilities	source program line continuation
U-36	using \$EDXASM L-4, U-361
secondary option menus S-218,	source statements, \$EDXASM overlay
U-36	generated I-243
storage usage \$-211	SP spool function,
terminal utilities U-40	\$RJE2780/\$RJE3780 reset type
text editing utilities U-36	C-76
utilities not supported U-46 SET,ATTN TERMCTRL function L-288	SPACE listing control statement coding description L-275
set breakpoints and trace ranges,	overview L-28
AT \$DEBUG command U-90	special control characters S-46
set date and time, \$T operator	special PI
command S-63, U-19	bit, SB \$IOTEST command U-267
SET Host Communications Facility	group, SG \$10TEST command
TP operand C-97	U-267
SET, LPI TERMCTRL function L-288	specifications, data conversion
set status, SE \$HCFUT1 command C-110	L-146
A TIA	

SPECPI define special process	strings, relational statement
interrupt L-189 SPECPIRT instruction	L-180 SU
coding description L-276	submit (X) function,
overview L-39	\$RJE2780/\$RJE3780 reset type
split screen configuration \$-293	C-77
SPOOL define spool file,	submit job to host, \$HCFUT1
\$RJE2780/\$RJE3780 C-76	command C-111
SQ set prompt made, \$COPYUT1	SUBMIT
command U-64	Host Communications Facility,
SQRT data manipulation instruction	TP operand C-98 send data stream to host,
coding description L-277	\$RJE2780/\$RJE3780 C-77
overview L-19	submit job to host, \$EDIT1
SS set program storage parameter,	command U-179
\$DISKUT2 command U-149	submit job to host, \$FSEDIT
ST	option U-217
display data set status,	SUBMITX send transparent,
\$DIUTIL command U-162	\$RJE2780/\$RJE3780 C-77
save disk or disk volume on tape, \$TAPEUT1 command U-330	SUBROUT program control statement coding description L-281
standard labels, tape	overview L-32, S-31
E0F1 S-240	subroutines
EOV1 S-239	\$IMDATA S-303
fields S-238	\$IMDEFN S-301
HDR1 S-239	\$IMOPEN S-300
header label 5-235	\$IMPROT S-302
layouts S-236	ALTIAM concatenation S-167 DSOPEN S-322
processing S-236 trailer label S-235	overview S-31
volume label S-235	SETEOD S-324
VOL1 S-238	SUBTRACT data manipulation
START	instruction
IDCB command L-176	coding description L-283
PROGRAM statement operand	overview L-19
L-225	precision table L-284
start and termination procedure, \$DEBUG U-85	suggested utility usage U-48 supervisor/emulator
STARTPGM I-16	class interrupt vector table
statement label L-4	I-10, I-277
static screen, terminal I/O	communications vector table
accessing example S-297	I-11, I-278, I-313
overview L-48	control block pointers I-11
status, set, SE \$HCFUT1 command	design features S-13
C-110 STATUS data definition statement	device vector table I-11, I-278
coding description L-278	emulator command table I-13,
overview L-17	I-282, I-301
status data set, system Host	entry routines I-47
Communications Facility C-85	equate table I-279, I-313
Status record C-258	exit routines_ I-49
STIMER timing instruction	features S-13
coding description L-280 overview L-50, S-32	fixed storage area I-9 functions I-44
with PASSTHRU function C-238	calling I-60
storage estimating	generation I-5, S-115
application program size	initialization control module
S-344	EDXINIT, description I-81
supervisor size \$-333	initialization task module,
utility program size S-342	EDXSTART, description I-81
storage management address relocation translator	interface routines I-61
I-71, S-42	introduction I-5 module names and entry points
allocating I-25	S-309
description S-42	module summary I-8
design feature S-13	overview S-29
storage map, resident loader I-26	PASSTHRU session with C-225
storage map (\$\$1ASM) phase to	referencing storage locations
phase I-262	in I-12
storage resident loader, RLOADER I-19	service routines I-53
	eign netimation Call
	size, estimating 5-333
storage usage during program load I-20	task supervisor work area
I-20 store next record (\$PDS) S-261	

ſ	_	т	_
_			

with the address translator	
support I-72	T
SUPEXIT supervisor exit routine I-49, I-63	
support for optional features	TA allocate tape data set,
L-15	\$TAPEUT1 command U-333
SUPRTURN supervisor exit routine	tables, parameter equate L-11
I-49	tabs
surface analysis of tape, \$TAPEUT1	HTAB \$IMAGE command U-252
utility U-319	TABSET \$EDIT1/N subcommand
SVC supervisor entry routine	U-198
I-47, I-62 SVCABEND supervisor exit routine	VTAB \$IMAGE command U-254 TABSET establish tab values
I-49	\$EDIT1/N editor subcommand U-198
SVCBUF supervisor request buffer	tape
I-48	bypass label processing S-244
SVCI supervisor entry routine	control L-74
I-48	data set L-40
symbol dictionary, \$EDXASM I-250	defining volumes S-62
symbol table types, \$EDXASM I-216 symbolic L-10	definitions for data sets L-40
address (disk, tape) L-10	end-of-tape (EOT) L-41
disk/tape I/O assignments	I/O instructions L-40
L-10	internals I-97
diskette L-10	labels
reference to terminals S-110	external 5-233
sensor I/O addresses L-9	internal S-233
terminal I/O L-10 symbols (EXTRN) L-134	load point (BOT) L-40 non-label
symbols (WXTRN) L-323	layout S-242
syntactical coding rules L-4	processing S-243
syntax checking in instruction	support S-241
parsing (\$EDXĀSM) I-221	record L-40
syntax rules L-4	return codes L-77, U-455
SYSGEN (see system generation)	standard label
system	fields S-238
alternate logging device S-46, S-111	layout S-236 processing S-236
class interrupt vector table	support S-235
I-10, I-277	storage capacity S-59
commands (see operator	symbolic addressing L-10
commands)	utility, \$TAPEUT1 S-233,
common area I-12	U-311
communications vector table I-11, I-278, I-313	volume L-40 TAPE configuration statement S-94
control blocks, referencing	tape data set control block I-99
I-289	tape device data block (see TDB)
data tables, EDXSYS, module	TAPEINIT, tape initialization mod-
description I-75	ule description I-82
device vector table I-11,	tapemark L-74
I-278	task
emulator command table I-13, I-282, I-301	active/ready level table I-50 concepts I-29
generation	control I-42
procedure S-115	control block (see TCB)
host/remote C-205	definition and control func-
logging device S-46, S-110	tions
operational and error	dispatching I-52
messages U-421	error exit facility
printer S-46, S-110 program check and error	check and trap handling S-268
messages U-427	linkage conventions S-269
task supervisor work area	execution states I-43, S-39
I-13, I-280	internals I-42
SYSTEM configuration statement	multiple-task program I-33,
L-39, S-86	5-34
system configuration statements	overview L-42, S-29
S-75 system control blocks S-42	priority (see priority, task execution)
system control blocks 5-42 system reserved labels L-4	single-task program I-33,
	S-34
	states S-39
	status display, WHERE \$DEBUG
	command U-102
	structure S-29

supervisor I-42	forms control L-45
supervisor address translator	prompting and advance
support module I-76	input L-46
supervisor functions I-44	screen management L-48
supervisor work area I-13,	I/O internal design I-105
I-280	I/O support layer 3 I-112
switching I-51, S-30	input L-46
synchronization and control	keyboard and ATTNLIST tasks
I-54, S-30	L-47
task code words L-8	message sending utility,
TASK task control statement	\$TERMUT3 U-344
coding description L-285	new I/O terminal support
overview L-42, S-31	I-117
	<del>-</del>
TASKSAVE supervisor service	operations C-14
routine I-54	operator signals L-49
TCB task control block I-32,	output L-46
I-43, I-49, I-56, I-314	output line buffering L-46
TCBEQU L-13	program function keys L-47
TD	prompting and advance input
display line and data (\$PDS)	L-46
5-258	return codes C-20, L-219,
display time and date, \$DICOMP	L-255, U-458
subcommand U-124	roll screens L-48
test display, \$DICOMP command	sample terminal support
U-108	program C-26
TDB, tape device data block	screen management_ L-48
description I-97	server, Multiple Terminal
equate listing I-316	Manager C-119, M-7
TEB terminal environment block	session manager (see session
C-128, M-13	manager)
Tektronix C-6	special considerations for
devices supported S-14, S-45	attachments of devices
support for digital I/O S-312	via #1610 or #2091 with
teleprocessing (see TP)	#2092 adapters C-17
teletypewriter adapter C-7, C-21	via #2095 with #2096
TERMCTRL terminal I/O instruction	adapters C-21
coding description L-288	special control characters
overview L-44	S-46
return codes L-301	static screens L-48
TERMERR L-44	supported devices and
terminal	features C-6
#7850 teletypewriter adapter	terminal I/O L-47
C-21	terminology for supported
ACCA support C-7, L-295	terminals C-7
ASCII C-7	transmission protocol C-31
assignment list, LA \$TERMUT1	utilities (session manager)
command U-336	S-215, U-40
attention handling L-47	virtual I/O I-115
attention keys L-47	TERMINAL configuration statement
code types C-303	defaults S-105
configuration utility,	definition S-96
\$TERMUT1 U-334	overview S-48
	TERMINAL volume, Multiple Terminal
connected via digital I/O	
5-312	Manager C-120, C-171
control block (see CCB)	terminate
data representation L-46	logging, \$LOG utility U-292
definition and control	Remote Management Utility
functions S-47	C-251
device configurations C-8	test
EDXTIO/EDXTIOU_module	BSC lines, \$BSCUT2 utility
description I-78	C-64
environment block (see TEB)	generated report or graphics
error handling L-44	profile member U-108
forms control L-46	label types, \$TAPEUT1 utility
forms interpretation for	U-319
display screens L-46	
	process interrupt for
functions	occurrence of event, \$IOTEST
data formatting C-16	U-267
definition C-16	TEXT data definition statement
definition C-16 interrupt processing C-17	
interrupt processing C-17	TEXT data definition statement
interrupt processing C-17 hardware jumpers C-18	TEXT data definition statement coding description L-305 overview L-17
interrupt processing C-17 hardware jumpers C-18 I/O L-46	TEXT data definition statement coding description L-305 overview L-17 text editing utilities
interrupt processing C-17 hardware jumpers C-18 I/O L-46 attention handling L-47	TEXT data definition statement coding description L-305 overview L-17 text editing utilities edit dataset subroutine exam-
interrupt processing C-17 hardware jumpers C-18 I/O L-46	TEXT data definition statement coding description L-305 overview L-17 text editing utilities

U-209 line editors, \$EDIT1/N U-169 overview S-66 work data set, format of text wrapping, WRAP function C-254 time/date display, \$W operator command U-25 set, \$T operator command U-19 set, automatic initialization facility S-130 time of day
GETTIME instruction L-167 TIMEDATE Host Communications Facility, TP operand C-100 TIMER configuration statement S-33, S-112 timer control L-50 timer module descriptions (EDXTIMER, EDXTIMR2) I-80 timing instructions L-50, S-32 TITLE listing control statement coding description L-308 overview L-28 TONE TERMCTRL function L-288 TOP repostiton line pointer, \$EDIT1/N editor subcommand U-200 TP host communication instruction description coding description C-90 internals I-153 subcommand operations I-157 TPCOM host communications support module description I-81 trace printing routine for BSC, \$BSCUT1 C-62, S-65 trace ranges and breakpoints setting, AT \$DEBUG command trace routine for BSC, \$BSCTRCE C-61 trace table, software S-265 transaction program, Multiple Terminal Manager functions L-28 Multiple Terminal Manager C-121 transfer data set to host SEND function C-247 WR \$HCFUT1 command C-112 WRITE \$EDIT1 command U-180 WRITE \$FSEDIT option U-216 transfer rates for data, Host Communications Facility transient program loader I-19 transmission codes S-98 transmission protocol, host communications I-156 transmitted data, length of, host communications I-159 TRAPDUMP force trap dump, \$TRAP attention command U-349 TRAPEND end \$TRAP use, \$TRAP attention command U-349 TRAPOFF deactivate error trap, \$TRAP attention command U-349 TRAPON activate error trap, \$TRAP attention command U-349

U

UN unload indexed file, \$IAMUT1 command U-246 UNBLINK TERMCTRL function L-288 undefined length records, tape S-245 UNLOCK TERMCTRL function L-288 unprotected field S-307, U-253 UP move line pointer, \$EDIT1/N editor subcommand U-201 update utility **\$UPDATE** convert object program to disk U-408 \$UPDATEH convert host object program to disk U-418 updating a menu for the session manager S-224 user defined data member (\$PDS) S-252 user exit routine L-310 requires Macro Assembler S-71 user initialization modules I-17 USER program control instruction coding description L-310 overview L-32 utilities U-47 BSC communications C-61 invoking U-2 listed by type S-64, U-3 overview S-5 utilities not supported by session manager menu U-46 utility program size S-342 utility usage U-48

٧

V verify, \$INITDSK command U-260 display, variable, \$DICOMP subcommand U-125 display variable (\$PDS) variable length record, Host Communications Facility C-84 variable length records, tape 5-244 variable names L-4 vary disk, diskette, or tape offline, \$VARYOFF U-20 vary disk, diskette, or tape online, \$VARYON U-22 vector addition L-19, L-54 data manipulation L-19 vector addition (ADDV) coding description L-54 overview L-19 verify disk or diskette data set, V \$INITDSK U-260 tape executing correctly, EX \$TAPEUT1 command U-319 tape surface free of defects, EX \$TAPEUT1 command U-319 verify and initialize disk or diskette library, \$INITDSK U-256 verify identification host system C-223 remote system C-223

359

VERIFY verify changes, \$EDIT1/N editor subcommand U-202 vertical tabs, defining U-VI list volume information, \$IOTEST command U-270 U-254 virtual terminal communications accessing the virtual termi-nal S-281 creating a virtual channel S-280 establishing the connection 5-280 inter-program dialogue S-282 internals I-115 loading from a virtual terminal S-281 Remote Management Utility requirements C-281 definitions (disk/diskette) L-22, S-52 dump restore utility, \$MOVEVOL U-294 labels S-60 VTAB define vertical tab setting, \$IMAGE command U-254

### M

WAIT program sequencing statement coding description L-313 overview L-42, S-31 supervisor function I-45, I-58 wait state, put program in, WS \$IOTEST command U-264 waiting, task execution state I-43 WE copy to basic exchange diskette data set, \$COPY command U-63 WHERE display status of all tasks, \$DEBUG command U-102 WHERES task control function coding description L overview L-42, 5-287 return codes L-316 WI write non-transparent, \$BSCUT2 command C-69 WIX write transparent, \$BSCUT2 command C-69 word boundary requirement DO L-34 IF L-34 PROGRAM L-225 work data set \$EDXASM I-249 \$LINK U-400 \$S1ASM I-258 work files, \$S1ASM, how used I-258 WR write a data set to host, \$HCFUT1 command C-112 WRAP function C-254, I-166, I-176 WRITE disk/diskette I/O instruction coding description L-317 overview L-22 return codes L-320, U-455 Host Communications Facility, TP operand C-101 IDCB command L-175 Multiple Terminal Manager

CALL coding description C-133, L-381 internals overview C-118, L-29 save work data set \$EDIT1 command U-180 \$EDIT1N command U-181 \$FSEDIT primary option U-216 tape I/O instruction coding description L-317 overview L-22 L-320, U-456 return codes L-320, U-456 write data set to host, WR \$HCFUT1 command C-112 write operations, HCF I-156 WRITE1 IDCB command L-175 WS put program in wait state, \$IOTEST command U-264 WTM (write tape mark) L-75 WXTRN program module sectioning statement coding description L-323 overview L-33

### XYZ

X-type format L-154
XI external sync DI, \$IOTEST command U-266
X0 external sync DO, \$IOTEST command U-266
XYPLOT graphics instruction coding description L-324 overview L-26
YTPLOT graphics instruction coding descrition L-325 overview L-26
ZCOR, sensor I/O L-189

#### Numeric Subjects

1560 integrated digital input/output non-isolated feature C-6 different device configurations C-8 use\_with different terminals C-7 1610 asynchronous communications single line controller C-6 considerations for attachment of devices C-17 different device configurations for interprocessor communications C-29 to a single line controller 5-99 use with different terminals C-7 2091 asynchronous communications eight line controller C-6, S-99 considerations for attachment of devices C-17 different device configurations C-8 use with different terminals

C-7	supported by Indexed Access
2092 asynchronous communications	Method S-146
four line adapter C-6	4969 Magnetic Tape Subsystem
considerations for attachment	S-233
of devices C-17	4973 Line Printer
different device	defined in TERMINAL configura-
configurations C-8	tion statement S-96
to attach ACCA terminal 5-99	end of forms S-307
use with different terminals	TERMINAL statement example
C-7	S-105
2095 feature programmable eight	4974 Matrix Printer
line controller C-6	defined in TERMINAL configura-
considerations for attachment	tion statement S-96
of devices C-21	end of forms \$-307
different device	restore to standard character
configurations C-8	set, RE \$TERMUT2 U-339
use with different terminals	TERMINAL statement example
C-7	S-105
	4978 Display Station
2096 feature programmable four	
line adapter C-6	defined in TERMINAL configura-
considerations for attachment	tion statement S-96
of devices C-21	part of minimum system
different device	configuration S-22
configurations C-8	reading modified data \$-307
use_with different terminals	required for program
C-7	_preparation S-22
2741 Communications Terminal	TERMINAL statement example
supported 5-45	S-105
TERMINAL statement example	4979 Display Station
S-106	defined in TERMINAL configura-
3101 Display Terminal	tion statement S-96
attribute character C-122	part of minimum system
block mode considerations	configuration S-22
C-25	required for program
character mode considerations	preparation S-23
C-22	TERMINAL statement example
interface with Multiple	S-105
Terminal Manager C-121, L-29	4982 sensor I/O unit S-84
TERMINAL configuration	5230 Data Collection Interactive
statement examples S-108	S-11
3585 4979 display station	5620 4974 matrix printer
attachment C-6, S-97	attachment C-6
4952 Processor	defined in TERMINAL statement
partitions on S-42	S-97
timer feature installed on	different device
5-32	configurations C-8
4953 Processor	5630 4973 line printer attachment
partitions on S-42	C-6
timer feature installed on	defined in TERMINAL statement
S-32	S-97
4955 Processor	5719-AM3 (see Indexed Access
partitions on S-42	Method)
timer feature installed on	5719-ASA (see Macro Assembler)
S-32	5719-CB3 (see COBOL)
4962 Disk Storage Unit	5719-CB3 (See COBOL)
storage capacity \$-58	5719-F02 (see FORTRAN IV) 5719-LM3 (see
supported by Indexed Access	
Method S-146	Mathematical/Functional Subrou-
4963 Disk Subsystem	tine Library)
storage capacity 5-58	5719-LM5 (see Macro Library)
supported by Indexed Access	5719-MS1 (see Multiple Terminal
Method S-146	Manager)
4964 Diskette Storage Unit	5719-SM2 (see Sort/Merge)
part of minimum system config-	5719-UT3 (see Utilities)
uration S-22	5719-UT4 (see Utilities)
required for program	5719-X\$1 (see Basic Supervisor and
preparation S-22	Emulator)
supported by Indexed Access	5719-XX2 (see Program Preparation
Method S-146	Facility)
4966 Diskette Magazine Unit	5740-LM2 (see Macro Library/Host)
part of minimum system config-	5799-TDE (see Data Collection
uration S-22	Interactive)
required for program	7850 teletypewriter adapter C-6,
	C-21

SC34-0316-2

## IBM Series/1 Event Driven Executive Communications and Terminal Applications Guide

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or the IBM branch office serving your locality.

Corrections or clarifications needed:

Page Comment

Please indicate your name and address in the space below if you wish a reply.

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.)

Fold and tape

Please Do Not Staple

Fold and tape

**Cut Along Line** 



**BUSINESS REPLY MAIL** 

FIRST CLASS

PERMIT NO. 40

ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation Systems Publications, Dept 27T P.O. Box 1328 Boca Raton, Florida 33432 NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



Fold and tape

Please Do Not Staple



SC34-0316-2

# IBM Series/1 Event Driven Executive Communications and Terminal Applications Guide

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or the IBM branch office serving your locality.

Corrections or clarifications needed:

Page Comment

Please indicate your name and address in the space below if you wish a reply.

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.)

\_ Cut or Fold Along Line \_

Fold and tape

Please Do Not Staple

Fold and tape



**BUSINESS REPLY MAIL** 

FIRST CLASS

PERMIT NO. 40

ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation Systems Publications, Dept 27T P.O. Box 1328 Boca Raton, Florida 33432 NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



Fold and tape

Please Do Not Staple



SC34-0316-2

# IBM Series/1 Event Driven Executive Communications and Terminal Applications Guide

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or the IBM branch office serving your locality.

Corrections or clarifications needed:

Page Comment

Please indicate your name and address in the space below if you wish a reply.

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.)

#### 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, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation Systems Publications, Dept 27T P.O. Box 1328 Boca Raton, Florida 33432



Fold and tape

Please Do Not Staple



SC34-0316-2

# IBM Series/1 Event Driven Executive Communications and Terminal Applications Guide

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or the IBM branch office serving your locality.

A		1	
Corrections	$\alpha$ r	clarifications	needed.
COLLECTIONS	O:	Ciai ii Ca dolla	necueu.

Page Comment

Please indicate your name and address in the space below if you wish a reply.

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.)

Fold and tape

Please Do Not Staple

Fold and tape



BUSINESS REPLY MAIL

FIRST CLASS

PERMIT NO. 40

ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation Systems Publications, Dept 27T P.O. Box 1328 Boca Raton, Florida 33432 NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



Fold and tape

Please Do Not Staple



