IBM

Series/1

SC34-0644-0

Event Driven Executive Operator Commands and Utilities Reference

Version 5.0

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

IBM

Series/1

SC34-0644-0

Event Driven Executive Operator Commands and Utilities Reference

Version 5.0

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

First Edition (December 1984)

Summary of Changes For Version 5.0

The following changes have been made to this document.

- The Job Cross-Reference Chart in *Chapter 1.* has been updated to include $TRANS, $TRAP, $DUMP, $MEMDISK, $COMPRES, and $E.
- A new utility, $TRANS has been added for Version 5. It replaces the CS command of $COPYUT1. This utility makes it possible to transport EDX data sets and programs from one Series/1 to another. Refer to "$TRANS - Transmit Data Sets Across a Bisync Line" on page UT-581 for this information.
- The section "$DASDI - Format Disk or Diskette" on page UT-90 has also been updated to reflect the ability to use option 0 to create both a stand alone dump and a $TRAP diskette.
- The $TRAP section now contains information on how to dump storage to multiple diskettes for systems larger than 512K. This information is located in "$TRAP - Save Storage on Error Condition" on page UT-589.
- Changes have been made to the prompt messages for the $DUMP utility. These changes are located in "$DUMP - Format and Display Saved Environment" on page UT-228.
- LIST and PATCH commands of the $DEBUG utility have been updated to illustrate their use with unmapped storage. A new CLOSE command and new prompts which appear when $DEBUG is loaded have also been documented in "$DEBUG - Debugging Tool" on page UT-126.
- Use of the $LOG utility is now required. Additional information regarding CIRCBUFF data is also contained in this version. Details on use of $LOG can be found in "$LOG - Log Errors into Data Set" on page UT-457.

Summary of Changes For Version 5.0

- Information about the 4975-01 ASCII printer is now included in the section entitled "$TERMUT1 - Change Terminal Parameters" on page UT-548.

About This Book

This book is a reference book containing detailed descriptions of the Event Driven Executive operator commands, session manager, and system utilities. The commands and applicable syntax for each operator command and system utility are shown, along with usage examples.

Audience

This book is intended for anyone who has to operate the IBM Series/1 with the Event Driven Executive. Readers should have a basic understanding of computer terminology before using this book.

How This Book Is Organized

The book is divided into the following 4 chapters.

- *Chapter 1, Introduction* contains an overview of the contents of the book.
- *Chapter 2, Operator Commands* contains a description and the syntax of the operator commands.
- *Chapter 3, Session Manager* contains a description of the session manager facility.
- *Chapter 4, Utilities* contains a description of each system utility used to operate your Event Driven Executive system. The utilities are presented in alphabetical order.

About This Book

Aids in Using This Book

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

- A chart of the main jobs that are done on the Series/1 with EDX. The chart is found under "Job Cross-Reference Chart" on page UT-2. It lists the operator command or system utility that is used for the job, the session manager option numbers to help you access the utility with the session manager, and the prefix of the guide in the EDX library that contains instructions for doing the job.
- A table of contents that lists the major headings in this book.
- An index of the topics covered in this book.
- A glossary that defines terms and acronyms used in this book and in other EDX library publications.

Illustrations in this book are enclosed in boxes. Many illustrations display output formats printed while using the Event Driven Executive. In those cases where the actual printer output exceeds the size of the box, the information is shown in a modified format.

Examples of display screens are also enclosed in a box representing the outline of a screen. A partial screen is indicated by the top or bottom of the screen. In examples where a response is required, the sample response is shown in red.

A Guide to the Library

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

Refer to the *Messages and Codes* manual for information about return codes issued by each utility described in this reference.

Contacting IBM about Problems

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

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

Contents

Contents

Contents

Contents

Figures

Figures

Chapter 1. Introduction

Operating your IBM Series/1 Event Driven Executive (EDX) system involves many different tasks, as follows:

- Installing the starter system
- Generating a tailored operating system
- Developing application programs
- Operating your system
- Determining hardware and/or software problems
- Customizing your system

To perform these tasks, you use the operator commands and system utilities.

Operator Commands

Operator commands are instructions that represent a request for action by your EDX system. When you enter an operator command, your EDX system performs the action specified by the operator command.

Chapter 2, "Operator Commands" contains a description and the syntax of each operator command, along with examples of its usage.

Introduction

Session Manager

The session manager is a collection of predefined screens called "menus" that you can use to access system utilities and application programs from a display station.

Chapter 3, "Session Manager" contains an introduction to the session manager and a description of the facility.

System Utilities

The system utilities are a set of programs that do everyday jobs on your Series/1. The system utilities are independent programs that can be run concurrently with other application programs or utilities.

Chapter 4, "Utilities" contains a description of each system utility. The utilities are presented in alphabetical order.

Job Cross-Reference Chart

The following chart directs you to the operator command or system utility that will help you do a particular job. There are four columns within the chart.

JOB This column lists specific jobs you may want to perform.

OPERATOR COMMAND/UTILITY
This column lists the name of the operator command or utility used to perform the job.

SESSION MANAGER OPTION
If a utility can be accessed through the session manager, the session manager option number is listed.

GUIDE INFORMATION
If instructions for doing a specific job are included in the *Operation Guide*, *Event Driven Executive Language Programming Guide*, *Communications Guide*, *Problem Determination Guide*, or *Installation and System Generation Guide*, the page prefix of the guide is listed.

Job Cross-Reference Chart *(continued)*

The prefix for each guide within the EDX library follows:

PG	Event Driven Language Programming Guide
CO	Communications Guide
CU	Customization Guide
IS	Installation and System Generation Guide
OP	Operations Guide
PD	Problem Determination Guide

Job	Operator Command/ Utility	Session Manager Option	Guide Information
Allocate a data set	$DISKUT1 $TAPEUT1	3.1 3.10	OP, IS
Allocate an H-exchange data set	$HXUT1	3.11	OP
Allocate a member in a graphics data base	$DICOMP	5.2	-
Allocate a volume	$INITDSK	3.7	OP, IS
Allocate a volume under a fixed head	$INITDSK	3.7	OP, IS
Analyze GPIB errors	$GPIBUT1	4.9	CO, PD
Analyze hardware errors	$DUMP	9.1	PD
Analyze program checks	$DUMP	9.1	PD
Analyze tape surface for defects	$TAPEUT1	3.10	—
Browse a source data set	$FSEDIT	1	—
Cancel a program	$C	—	OP
Change hard-copy device	$TERMUT1	4.1	OP
Change print screen PF key	$TERMUT1	4.1	OP
Change spool job attributes	$SPLUT1	4.7	OP
Change a tape label	$TAPEUT1	3.10	OP

Figure 1 (Part 1 of 7). Job Cross-reference Chart

Job Cross-Reference Chart *(continued)*

Job	Operator Command/ Utility	Session Manager Option	Guide Information
Change a terminal address	$TERMUT1	4.1	OP
Close job created by $DEBUG	$DEBUG	—	—
Compile an EDL program	$EDXASM	2.1	PG, OP
Compile and link edit an EDL program	$EDXASM $EDXLINK	2.2	PG, OP
Compile a program on a S/370 and execute it on a Series/1	$UPDATEH	2.10	—
Compress one volume or all volumes on a device	$COMPRES	3.4	OP
Controlling the job queue processor	$JOBQUT	10.1	OP
Control printer spooling	$SPLUT1	4.7	OP
Convert object modules to executing code	$EDXLINK $UPDATE	2.7 2.9	PG
Copy all data sets	$COPYUT1	3.3	OP
Copy a data set with automatic allocation	$COPYUT1	3.3	IS, OP
Copy all data sets with the same prefix	$COPYUT1	3.3	OP
Copy a basic exchange data set	$COPY	3.5	OP
Copy a disk/diskette data set to tape	$TAPEUT1	3.10	OP
Copy an H-exchange data set	$HXUT1	3.11	OP
Copy a tape data set to diskette	$TAPEUT1	3.10	OP
Copy a tape data set to tape	$TAPEUT1	3.10	OP
Copy a $TRAP data set to two diskettes	$COPY	3.5	OP
Copy a volume or data set to an allocated volume or data set	$COPY	3.5	OP
Create an upper/lower case data set	$FSEDIT $TERMUT2	1 4.2	OP
Create your own terminal keyboard characters	$FONT	4.5	—
Date and time, display	$W	—	OP
Date and time, set	$T	—	OP

Figure 1 (Part 2 of 7). Job Cross-reference Chart

Job Cross-Reference Chart *(continued)*

Job	Operator Command/ Utility	Session Manager Option	Guide Information
Delete job queue	$JOBQUT	10.1	OP
Delete a data set	$DISKUT1	3.1	OP
Delete all data sets with the same prefix	$DISKUT1	3.1	OP
Delete a volume	$INITDSK	3.7	OP
Determine address of a data set	$DISKUT1	3.1	—
Determine how many records a data set contains	$DISKUT1	3.1	OP
Determine terminal name, address, type and partition assignment	$TERMUT1	4.1	OP
Determine how much free space is on a volume	$DISKUT1	3.1	OP
Determine the hardware supported by supervisor	$IOTEST	9.3	OP, IS
Device, set offline	$VARYOFF	—	OP
Device, set online	$VARYON	—	OP
Display data/volume/storage contents in decimal, EBCDIC, or hexadecimal	$DEBUG $DISKUT2	— 3.2,9.2	PD, OP
Dump storage to a data set on error condition	$TRAP	—	PD, OP
Dump unmapped storage	$TRAP $DUMP	—	PD, OP
Estimate progress of compress	$COMPRES	3.4	OP
Eject a page	$E	—	OP
End job queue processing	$JOBQUT	10.1	PD
Enter source statements	$FSEDIT	1	PG, OP
Erase contents of a data set	$DISKUT2	3.2,9.2	—
Erase display screen	$B	—	OP
Format and print BSC trace data sets	$BSCTRCE $BSCUT1	8.1 8.2	CO, PD
Format and print error log information	$DISKUT2	3.2,9.2	OP, PD
Generate a supervisor	$XPSLINK	2.8 2.13	IS

Figure 1 (Part 3 of 7). Job Cross-reference Chart

Job Cross-Reference Chart *(continued)*

Job	Operator Command/ Utility	Session Manager Option	Guide Information
Hold a job in the job queue processor	$JOBQUT	10.1	OP
Identify the program function keys on a 4978 and 4980 display station	$PFMAP	4.6	—
Increase the size of a data set	$DISKUT1 $COPY	3.1 3.5	—
Initialize a disk	$DASDI $MEMDISK	3.6 —	IS, OP
Initialize unmapped storage as a disk	$MEMDISK	—	OP
Initialize a diskette to EDX format	$DASDI	3.6	OP
Initialize a diskette to exchange format	$DASDI	3.6	OP
Initialize a diskette for a stand-alone/$TRAP dump	$DASDI	3.6	OP
Initialize a tape	$TAPEUT1	3.10	OP
Invoke your own operator command	$U	—	OP, CU
IPL a remote Series/1	$S1S1UT1	4.8	CO
Job queue processing controller	$JOBQUT	10.1	OP
Job queue job submission utility	$SUBMIT	10.2	OP
Link edit programs	$EDXLINK	2.7	PG, OP
List all data sets	$DISKUT1	3.1	OP
List data sets by type (program or data)	$DISKUT1	3.1	OP
List all data sets with same prefix	$DISKUT1	3.1	OP
List a directory	$INITDSK	3.7	OP
List all volumes on all disk/diskette devices	$DISKUT1 $INITDSK	3.1	OP
List all volumes on a particular device	$INITDSK	3.7	OP
List information about one data set	$DISKUT1	3.1	OP
List messages in source message data set	$MSGUT1	2.14	OP
Load and execute a program	$L	—	OP

Figure 1 (Part 4 of 7). Job Cross-reference Chart

Job Cross-Reference Chart *(continued)*

Job	Operator Command/ Utility	Session Manager Option	Guide Information
Load 4980 terminal	$TERMUT2	4.2	OP
Loading control image	$TERMUT2	4.2	OP
Loading control store	$TERMUT2	4.2	OP
Locate required data sets/overlays before program execution	$PREFIND	2.11	—
Message-source processing	$MSGUT1	2.14	OP
Modify a source data set	$FSEDIT	1	OP,PG
Partition assigned to terminal, change	$CP	—	OP
Prepare Version 1 and 2 data sets for use with Version 5	$MIGRATE $MIGAID $MIGCOPY	—	IS
Print a formatted $TRAP dump	$DUMP	9.1	OP, PD
Print a stand-alone dump	$DUMP	9.1	OP, PD
Programs, display all active	$A ALL	—	OP
Programs, display all active in terminal's partition	$A	—	OP
Recognizing intermittent program errors	$TRAP	—	OP, PD
Recording hardware errors	$LOG	—	PD
Recover a backed-up data set	$MOVEVOL	3.8	OP
Redefine terminal scrolling	$TERMUT1	4.1	—
Rename a disk/diskette volume	$INITDSK	3.7	OP
Rename a terminal	$TERMUT1	4.1	OP
Restart job queue processing	$JOBQUT	10.1	OP
Retrieve a data set from a host	$EDIT1N	—	—
Running batch job streams	$JOBUTIL $SUBMIT	7 10.2	OP
Running multiple jobs	$JOBUTIL $SUBMIT	7 10.2	OP

Figure 1 (Part 5 of 7). Job Cross-reference Chart

Job Cross-Reference Chart *(continued)*

Job	Operator Command/ Utility	Session Manager Option	Guide Information
Send a data set to a host	$HCFUT1	8.8	CO
Send a message to a terminal	$TERMUT3	4.3	OP
Setting PF keys	$TERMUT2	4.2	OP
Setting a terminal offline or online	$TERMUT1	4.1	OP
Sort a disk/diskette volume directory	$DIRECT	—	—
Spooling, change forms type	$S ALT	—	OP
Spooling, change heading id	$S ALT	—	OP
Spooling, change number of copies	$S ALT	—	OP
Spooling, change printer	$S ALT	—	OP
Spooling, delete a job	$S DEL	—	OP
Spooling, delete a set of jobs	$S DG	—	OP
Spooling, display status of jobs	$S DISP	—	OP
Spooling, display status of resources	$S DISP	—	OP
Spooling, display status of writers	$S DISP	—	OP
Spooling, hold jobs	$S HOLD	—	OP
Spooling, keep a job after printing	$S KEEP	—	OP
Spooling, release held jobs	$S REL	—	OP
Spooling, restart a writer	$S WRES	—	OP
Spooling, start	$L $SPOOL	—	OP
Spooling, start a writer	$S WSTR	—	OP
Spooling, stop	$S STOP	—	OP
Spooling, stop a writer	$S WSTP	—	OP
Storage, change contents (patch)	$P	—	—

Figure 1 (Part 6 of 7). Job Cross-reference Chart

Job Cross-Reference Chart *(continued)*

Job	Operator Command/ Utility	Session Manager Option	Guide Information
Storage, display contents	$D	—	OP
Storage map, display for all partitions	$A ALL	—	OP
Storage map, display for terminal's partition	$A	—	OP
Submit a job into a host batch job stream	$EDIT1N	—	—
Suspend job queue processing	$JOBQUT	10.1	OP
Test the binary synchronous access method (BSCAM)	$BSCUT2	8.3	CO, PD
Test tape label type to ensure I/O commands executing correctly	$TAPEUT1	3.10	—
Test the operation of sensor I/O features	$IOTEST	9.3	—
Trace I/O activities on a BSC line	$BSCTRCE	8.1	CO, PD
Transport data sets across a bisync line	$TRANS	—	OP
Trap storage image on error condition	$TRAP	—	PD
Trap to two diskettes	$TRAP	—	PD, OP
Verify BSC hardware and software assignments	$BSCUT2	8.3	CO, PD
Verify Series/1-Series/1 attachment performing correctly	$S1S1UT1	4.8	CO, PD
Write a tape label	$TAPEUT1	3.10	OP

Figure 1 (Part 7 of 7). Job Cross-reference Chart

Notes

Chapter 2. Operator Commands

The system operator commands provide system control functions from your terminal. They tell EDX to do things such as load a program and set the time and date. The 14 operator commands and their functions are:

Command	**Function**
$A	Displays partition sizes and addresses and the names, locations, and loading terminal names of all loaded programs.
$B	Blanks the display terminal screen.
$C	Cancels a running program.
$CP	Changes the partition assigned to a terminal.
$D	Displays the contents of storage.
$E	Ejects a page on the printer.
$L	Loads a program.
$P	Patches storage.
$S	Controls the spooling of program output.
$T	Enters the date and time.
$U	Invokes a user-written routine.

$VARYOFF	Sets a device offline.
$VARYON	Sets a device online.
$W	Displays the date and time.

This chapter shows you how to enter an operator command and describes the function and syntax of each command. Refer to the *Operation Guide* for procedures that use the operator commands.

Entering Commands

You can enter operator commands in one of two ways: *prompt-reply* or *single-line* format. With prompt-reply format you enter the command name and each parameter as the system asks for it. With single-line format you enter the command name and all the parameters on the same line. The following examples show you how to use the two formats.

Prompt-Reply Format

Press the attention key. After EDX responds with the greater-than sign (>), type the operator command and press the enter key. EDX responds with a prompt for the next parameter as each parameter is entered.

```
> $L
 PGM(NAME,VOLUME,STORAGE): MYPROG,MYVOL,256
 LOADING MYPROG       4P,08:30:45, LP- 0000, PART=2
```

Single-Line Format

Press the attention key. After EDX responds with the greater-than sign (>), type the operator command and all parameters in the order expected by EDX, and press the enter key.

You can enter all operator commands (except $T - set date and time) in the single-line format.

```
> $L MYPROG,MYVOL,256
```

Operator Command Descriptions

This section contains a description and the syntax of each operator command arranged in alphabetical order.

Syntax Conventions

The following conventions are used in the presentation of the operator command syntax:

Uppercase	If a parameter is shown in all uppercase letters, enter it exactly as shown.
Lowercase	If a parameter is shown is all lowercase letters, substitute a variable value.
[ ]	If two or more parameters are enclosed in brackets, chose one of them. The parameters within the brackets are separated by the *or* (|) character.
|	If parameters are separated by the *or* (|) character, choose one of them.

$A - List Partitions and Active Programs

Use the $A operator command to list the storage partitions defined for your EDX operating system. (A partition is a portion of storage in which programs run.) When you enter the $A command, the system provides the starting address, the name of the terminal from which the program was loaded, the size of each partition, and whether or not a partition is static or dynamic. It also tells you the names and address of each program active in each partition.

You can use $A to list *all* partitions or only the one to which your terminal is assigned.

Syntax:

```
$A          ALL|blank

Required:   none
Default:    starting address,size, and active programs of
            partition currently assigned to
```

Operator Command Descriptions *(continued)*

Operands	*Description*
ALL	Displays the starting addresses and sizes of all partitions and the active programs in each partition.
blank	If you do not enter ALL, EDX displays the starting addresses of the programs active in the partition where your terminal is running.

$B - Blank Display Screen

Use the $B operator command to erase all information on your display terminal screen.

Syntax:

```
$B

 Required:   none
 Default:    none
```

Operands	*Description*
None	None

$C - Cancel Program

Use the $C operator command to cancel a program running in the same partition as your terminal. If there is more than one program of the same name, EDX asks (prompts) you for the load address of the program. The load address is the storage address where the program starts. You can find this address using the $A operator command.

Notes:

1. $C should not be used as the normal means of stopping a program.
2. $C should not be used to cancel some of the system utilities. If $C should not be used, the utility warns you on the first screen it displays.

Syntax:

```
$C          program

Required:   program
Default:    none
```

Operands	*Description*
program	The name of the program to cancel.

$CP - Change Display Terminal's Partition Assignment

Some jobs require that the display terminal be running in the same partition as the object of the job, such as $C to cancel a program. Use the $CP operator command to change the partition number for the terminal you are using.

Syntax:

```
$CP         n

Required:   n
Default:    none
```

Operands	*Description*
n	The partition to which the terminal is to be assigned.

Operator Command Descriptions *(continued)*

$D - Dump Storage

Use the $D operator command to display, or *dump*, the contents of an area of storage on the screen of your display terminal. When you enter the $D command, EDX displays the hexadecimal contents of the specified storage locations.

Syntax:

```
$D          origin,address,count

 Required:   origin
 Default:    address defaults to 0
             count defaults to 1.
```

Operands	***Description***
origin	The hexadecimal origin address. This can be any address, but if the address parameter is a displacement into a program, this value is the program load point.
address	The hexadecimal displacement from origin at which the dump is to start.
count	The decimal number of words to dump (maximum value of 16).

$E - Eject Printer Page

Use the $E operator command to advance (eject) one or more pages on the specified printer. Entering a number with $E advances the paper that number of pages.

Syntax:

```
$E          n printername

 Required:   none
 Default:    ejects one page, $SYSPRTR
```

Operands	***Description***
n	The number of pages to eject. Defaults to printername.

$L - Load a Program or Utility

Use the $L operator command to load a program into storage and start it running.

Syntax:

$L	program,volume,storage data sets
Required:	program
Default:	volume defaults to **IPL** volume; storage defaults to the amount specified on the **PROGRAM** statement of the program to be loaded

Operands	***Description***
program	The name of the program being loaded. This is the same as the name of the data set where the program is stored.
volume	The name of the volume containing the program data set.
storage	The total additional storage (in bytes) to be added to the end of the loaded program (overrides the STORAGE= parameter specified in the PROGRAM statement). The number of bytes that must be specified, if any, is determined by local procedures. If a '*' is specified, the loaded program will receive the maximum amount of contiguous free space available in that partition.
data sets	The data set and volume names of one to nine data sets being passed to the program. This parameter is required if DS=?? is coded on the PROGRAM statement. The data sets must be specified in the order used by the program and entered in the format: *name,volume name,...* If data set names are required and you do not enter them as part of a single-line command, EDX prompts you for them.

Operator Command Descriptions *(continued)*

$P - Patch Storage

Use the $P operator command to change (patch) one or more words of storage. (Refer to the *Problem Determination Guide* for instructions on using $P.)

Note: Patching of main storage is only valid for the current session. When the system is reinitialized (IPL'D) or the executing program is reloaded, the patched data reverts to its original value.

Syntax:

```
$P          origin,address,count

 Required:   origin
 Default:    address defaults to 0
             count defaults to 1.
```

Operands	***Description***
origin	The hexadecimal origin address (program load point). Use the $A operator command to determine this address.
address	The hexadecimal address in the program where the patch starts.
count	The decimal number of words being patched. A maximum of 16 words can be patched.

$S - Control Spooled Program Output

Use the $S operator command to control the operation of printer spooling from your display terminal. $S has several subcommands that do these control functions. The syntax for these subcommands is described on the following pages under:

- "$S - List Subcommands" on page UT-20
- "$S ALT - Alter Spool Job Printing" on page UT-20
- "$S DALL - Delete All Spool Jobs" on page UT-21
- "$S DE - Delete a Spool Job" on page UT-22
- "$S DG - Delete Generic Spool Jobs" on page UT-22
- "$S DISP - Display Spool Status Information" on page UT-23
- "$S HOLD - Hold Spool Job(s)" on page UT-23
- "$S KEEP - Keep or Release a Spool Job" on page UT-24
- "$S REL - Release Spool Job(s)" on page UT-24
- "$S STOP - Stop Spooling Facility" on page UT-25
- "$S WRES - Restart a Spool Writer" on page UT-25
- "$S WSTP - Stop a Spool Writer" on page UT-26
- "$S WSTR - Start a Spool Writer" on page UT-27

Operator Command Descriptions *(continued)*

$S - List Subcommands

Use the $S command to obtain a list of the $S subcommands.

Syntax:

```
$S

 Required: none
 Default: none
```

$S ALT - Alter Spool Job Printing

Use the $S ALT command to change the parameters that control the way a spool job is printed. You can:

- Change the number of copies printed.
- Change the forms code for the job.
- Change the job name used on the spool job separator page.
- Redirect one spool job to a different printer.
- Redirect all spool jobs from one printer to another printer.
- Specify that forms alignment be verified before a job is printed.

Syntax:

```
$S  ALT   id    [COPY n|FORM code|NAME heading|WRIT name|ALIGN Y/N]
           or
$S  ALT   WRIT   cwriter nwriter

 Required: id and either COPY FORM NAME WRIT or ALIGN
           or
            WRIT cwriter nwriter
 Default:  none
```

Operands	*Description*
id	The one-to-three-digit identification assigned to a spool job by the spool facility. This identification is included on the spool status report generated by the $S DISP ALL operator command.
COPY	n - The number of copies to be printed (must be from 1 to 127).
FORM	code - The four-character code identifying the forms required to print the spool job.
NAME	heading - A one-to-eight-character heading printed on the spool job separator page. It defaults to the name of the program which created the spool job.
WRIT	The name of the spoolable printer or display terminal. The "WRIT name" form of this parameter is used to assign a printer or display terminal to a particular spool job. The "WRIT cwriter nwriter" form is used to redirect spool jobs from one spool device to another.

name	The name of the spool device for this spool job.
cwriter	The name of the current spool device whose spool jobs are to be redirected.
nwriter	The name of the new spool device.

ALIGN	Y/N - Specifies whether forms alignment is to be verified before the spool job is printed (Y=yes, N=no). Alignment is verified for the next complete copy of the job. Alignment is not verified for a job that is printing when this command is entered or for a job that has been stopped with the $S WSTP command.

$S DALL - Delete All Spool Jobs

Use $S DALL to delete all ready or printing spool jobs.

Syntax:

```
$S  DALL

Required: None
Default:  None
```

Operands	*Description*
None	None

Operator Command Descriptions *(continued)*

$S DE - Delete a Spool Job

Use $S DE to delete one spool job that is either ready or printing.

Syntax:

```
$S  DE    id

 Required: id
 Default:   None
```

Operands	*Description*
id	The one-to-three-digit identification assigned to a spool job by the spool facility. This identification is included on the spool status report generated by the $S DISP ALL operator command.

$S DG - Delete Generic Spool Jobs

Use the $S DG command to delete all ready or printing spool jobs that have a name starting with a specified prefix.

Syntax:

```
$S  DG    string

 Required: string
 Default:   None
```

Operands	*Description*
string	A one-to-eight-character prefix that specifies the spool jobs to be deleted. All spool jobs with this prefix are deleted.

Operator Command Descriptions *(continued)*

$S DISP - Display Spool Status Information

Use $S DISP to display information about spool jobs, spool resources, and spool writers.

Syntax:

```
$S DISP     id|ALL|STAT

Required:   None
Default:    ALL
```

Operands	*Description*
id	The internal one-to-three-character identification assigned to a spool job by the spool facility. This identification is obtained by using the $S DISP ALL command.
ALL	Displays the status of all spool jobs, all spool writers, and all spool resources.
STAT	Displays the status of the spool resources.

$S HOLD - Hold Spool Job(s)

Use the $S HOLD command to hold a specific spool job, or all spool jobs, from being printed. Only active and ready spool jobs can be held.

Syntax:

```
$S HOLD     id|ALL

Required: None
Default:  ALL
```

Operands	*Description*
id	The one-to-three-character identification assigned to a spool job by the spool facility. This identification is included on the spool status report generated by the $S DISP ALL operator command.
ALL	Holds all active and ready spool jobs and all future spool jobs.

Operator Command Descriptions *(continued)*

$S KEEP - Keep or Release a Spool Job

Use $S KEEP to keep a specific spool job from being deleted or to delete a job that has been kept. When a kept job is released, $SPOOL prints one additional copy before deleting the job.

Syntax:

```
$S KEEP      id Y|N

 Required: id and either Y or N
 Default:  None
```

Operands	*Description*
id	The one-to-three-character identification assigned to a spool job by the spool facility. This identification is included on the spool status report generated by the $S DISP ALL operator command.
Y	Keeps the spool job available after it is printed. The spool job is both held and kept after printing, with the number of copies set to one. Thus, when released by using the $S REL command, it is printed (even if printed once already).
N	Deletes the spool job from the system after it is printed. When released by use of this operand, the number of copies of the spool job to be printed is set to one, even if more than one copy was requested before the job was kept.

$S REL - Release Spool Job(s)

Use $S REL to release one, or all, held jobs for printing. A released job resumes its place in the ready queue; that is, its print order is still determined by the order in which it originally became ready.

Syntax:

```
$S REL  id|ALL

 Required: None
 Default:  ALL
```

Operands	*Description*
id	The one-to-three-character identification assigned to a spool job by the spool facility. This identification is included on the spool status report generated by the $S DISP ALL operator command.
ALL	Releases all currently held spool jobs. This resets the effect of the $S HOLD ALL command.

$S STOP - Stop Spooling Facility

Use $S STOP to stop the spooling facility. Spooling stops when any jobs in active or printing status finish.

Syntax:

```
 $S  STOP

Required: None
Default:  None
```

Operands	*Description*
None	None

$S WRES - Restart a Spool Writer

Use $S WRES to restart a temporarily stopped spool writer. You can restart a writer:

- at the beginning of the interrupted job
- at the line following the last line printed
- at a specified number of lines or pages before or after the last line printed

Syntax:

```
 $S  WRES   name   [IMM|JOB|B nnn L/P|F nnn L/P] code

Required:   name
            nnn and L or P if B or F specified
Default:    IMM
            code defaults to blanks
```

Operator Command Descriptions *(continued)*

Operands	*Description*
name	The name of the writer which is to be restarted.
IMM	Resume printing at the *next line* of the interrupted spool job.
JOB	Resume printing at the *start* of the interrupted spool job.
B	Resume printing *nnn* lines (L) or pages (P) *before* the line where the job was stopped.
F	Resume printing *nnn* lines (L) or pages (P) *after* the line where the job was stopped.
nnn	The number of lines or pages backward (B) or forward (F) from the point of interruption. Specify this parameter if you specified B or F. If the writer scrolls to the start of the spool job, the complete job is printed. If the writer scrolls to the end of the spool job, the job is not printed.
L	Specifies that *nnn* is in lines.
P	Specifies that *nnn* is in pages.
code	The four-character forms code for the forms to be used for jobs printed by this writer. A new forms code specification takes effect when the writer processes the next job or the next copy of the current job. Defaults to blanks.

$S WSTP - Stop a Spool Writer

Use $S WSTP to stop a spool writer. You can stop a spool writer:

- immediately at the start of the next line of the spool job or at the end of the current job
- temporarily, to be restarted with the $S WRES command, or permanently. A spool job that is permanently stopped is restarted with the $S WSTR command.

Syntax:

```
$S WSTP   name   [IMM|JOB] [TERM|NOTERM]

 Required:   name
 Default:    IMM  NOTERM
```

Operands	*Description*
name	The name of the writer which is being stopped.
IMM	Stop printing at the next line of the spool job.
JOB	Stop printing at the end of the current job.
TERM	Stop permanently. (The dedicated printer is released and writer task is ended.) **Note:** A permanently stopped writer task can only be restarted by the $S WSTR command.
NOTERM	Stop temporarily. The writer can be restarted by the $S WRES command. (The writer maintains control of the printer device.)

$S WSTR - Start a Spool Writer

Use $S WSTR to start a spool writer and specify a forms code.

Syntax:

```
$S  WSTR   name code

Required:   name
Default:    code - defaults to blanks
```

Operands	*Description*
name	The name of the printer for which the writer is to be started. This is also the name of the writer.
code	The four-character forms code for the printer forms used with the jobs printed by this writer. Defaults to blanks.

$T - Set Date and Time

Use the $T operator command to set the date and time in the EDX operating system.

You can only enter $T from a display terminal named $SYSLOG or $SYSLOGA (the system logging display terminal or its alternate). If you enter it from a terminal with another name, the system displays the date and time as if you had entered a $W command.

Note: $T must be entered in prompt-reply format.

Operator Command Descriptions *(continued)*

Syntax:

```
$T          date,time

 Required:   date,time
 Default:    date defaults to 00/00/00
             time defaults to 00:00:00
```

Operands	***Description***
date	The current date. Can be entered as *mm/dd/yy*, *mm.dd.yy.*, or *mm dd yy*.
time	The current time. Can be entered as *hh:mm:ss*, *hh.mm*, or *hh mm*.

$U - Invoke Your Own Operator Command

Use the $U operator command to enter an operator command function that is unique to your system. Refer to *Customization Guide* for information on how to write a program for the $U operator command.

If $U is entered and your system does not have a program to support it, EDX displays the message "FUNCTION NOT DEFINED."

$VARYOFF - Set Device Offline

Use the $VARYOFF command to set a diskette or tape drive offline. When a diskette or tape drive is offline, the computer does not control it. When you vary a tape drive offline, the system rewinds the tape to the beginning.

When you remove a diskette from a diskette unit, issue the $VARYOFF command to vary the slot used offline. Otherwise, EDX will continue to use that diskette.

Syntax:

```
$VARYOFF   ioda slot

 Required:   ioda
 Default:    none
```

Operands	*Description*
ioda	The hexadecimal device address of the diskette or tape device being varied offline.
slot	The number of the 4966 diskettes being varied offline. This parameter is valid *only* for the 4966 diskette unit. The valid slot numbers for the 4966 magazine unit are:

0	All diskettes (1,2,3,A,B)
1	Slot 1
2	Slot 2
3	Slot 3
A	Magazine 1
B	Magazine 2

$VARYON - Set Device Online

Use the $VARYON operator command to set a diskette or tape drive online. Use the $VARYOFF operator command to cancel each $VARYON command.

You do not have to enter $VARYON when you put a new diskette into a 4964 or 4965 diskette unit. Your system automatically varies these devices online when you shut the door of the diskette unit. You MUST, however, issue the $VARYON command before attempting to use a tape unit or a 4966 diskette unit.

Syntax:

```
$VARYON    ioda slot|file EX

Required:  ioda
Default:   file defaults to 1
           maximum value of file is 255
```

Operands	*Description*
ioda	The hexadecimal device address of the diskette or tape device being varied online.
slot	The number of the 4966 slot containing the diskette being varied online. This parameter is valid only for the 4966 diskette unit. The valid slot numbers for the 4966 magazine unit are:

0	All diskettes (1, 2, 3, A, B)
1	Slot 1
2	Slot 2
3	Slot 3
A	Magazine 1

Operator Command Descriptions *(continued)*

	B Magazine 2
file	The decimal file number on the tape being accessed.
EX	Override the tape expiration date. If a tape data set is initialized with an expiration date, EX must be used to be able to write to that tape data set. The file number must also be specified.

The "file" and "EX" parameters are valid only for tape devices.

$W - Display Date and Time

Use the $W operator command to display the date and time, according to your EDX system, on your display terminal.

Syntax:

```
$W

 Required:   none
 Default:    none
```

Operands *Description*

None

Chapter 3. Session Manager

The session manager provides access to system utilities and application programs from a display terminal. It uses a series of menu screens to direct you to the system utility you need and/or prompts you for parameters, such as data set names, needed by the option you chose.

This chapter explains the session manager screens and options. It also contains a table that cross references the system utilities supported by the session manager to the appropriate menu option.

Session Manager

Invoking the Session Manager

The session manager must be active at your display terminal before you can use it. This can be accomplished by either loading it for that specific terminal, or having it loaded automatically during initial program load (IPL) of the EDX system. When the session manager is loaded during IPL, EDX loads a copy for each display terminal recognized by the operating system.

The session manager is loaded for a specific terminal using the $L operator command as follows:

```
> $L $SMMAIN
```

To load the session manager automatically, you must rename the session manager initialization program from $SMINIT to $INITIAL. This is done using the $DISKUT1 RE command as follows:

```
COMMAND (?): RE $SMINIT $INITIAL
```

When the session manager is loaded, it displays the logon menu shown in Figure 2 on page UT-33.

Menus

The session manager menus, or display screens, list system facilities available through the session manager. They also display prompts for required parameters.

The session manager has the following menus:

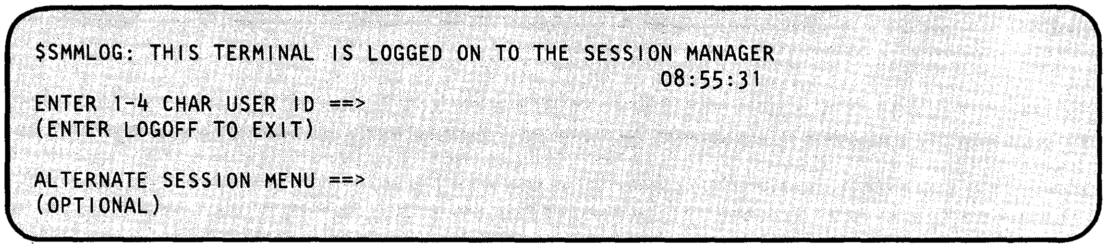
- Logon/logoff
- Primary option
- Secondary option
- Parameter input
- Custom

Logon/Logoff Menu

The logon menu prompts you for a user ID and an optional alternate session menu if you are logging on to the session manager, or for the word LOGOFF if you are ending your session (logging off).

Your user ID must be 1 to 4 unique characters, such as your initials. The session manager uses your ID as part of the data set names of six work data sets that it allocates for your session. It does not use your ID as a password to verify that you are authorized to use the system.

The alternate session menu is an alternate menu that you want displayed instead of the primary option menu. An alternate session menu is available only if your copy of the session manager has been customized. (Refer to the *Customization Guide* for instructions on adding menus to the session manager.)



```
$SMMLOG: THIS TERMINAL IS LOGGED ON TO THE SESSION MANAGER
                                                   08:55:31
ENTER 1-4 CHAR USER ID ==>
(ENTER LOGOFF TO EXIT)

ALTERNATE SESSION MENU ==>
(OPTIONAL)
```

Figure 2. Session manager logon/logoff menu

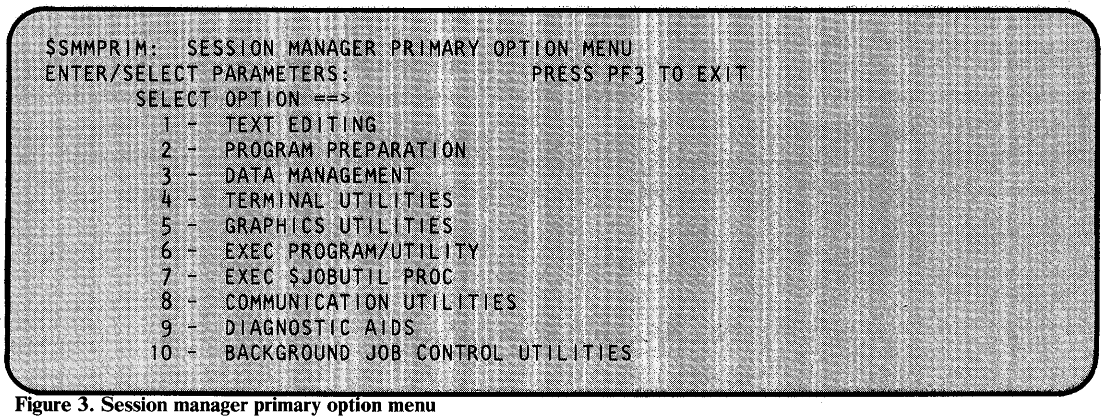
Primary Option Menu

The primary option menu lists all of the primary options provided with the session manager. If your session manager has been customized, you may have additional options, or the options may be different from the ones listed below. To select an option, enter the number of the option on the SELECT OPTION prompt line. After you select a primary option, the session manager displays a secondary option or parameter input menu. (See Figure 3 on page UT-34 for an example of the primary option menu.)

The basic options are:

1. TEXT EDITING: Accesses the $FSEDIT text editor.
2. PROGRAM PREPARATION: Accesses the program preparation utilities.
3. DATA MANAGEMENT: Accesses the utilities for managing data on disk, diskette, or tape.
4. TERMINAL UTILITIES: Accesses the terminal support utilities.
5. GRAPHICS UTILITIES: Accesses the utilities that generate, maintain, and display two- and three-dimensional fixed graphic backgrounds, and store them in data sets.
6. EXEC PROGRAM/UTILITY: Allows you to load any program. The program can be an EDX system program, an EDX utility, or an application program.
7. EXEC $JOBUTIL PROC: Allows you to load a previously built $JOBUTIL procedure.
8. COMMUNICATION UTILITIES: Accesses the utilities that support communications.
9. DIAGNOSTIC AIDS: Accesses the utilities that help with problem determination.
10. BACKGROUND JOB CONTROL UTILITIES: Accesses the job queue processing utilities.

Menus *(continued)*



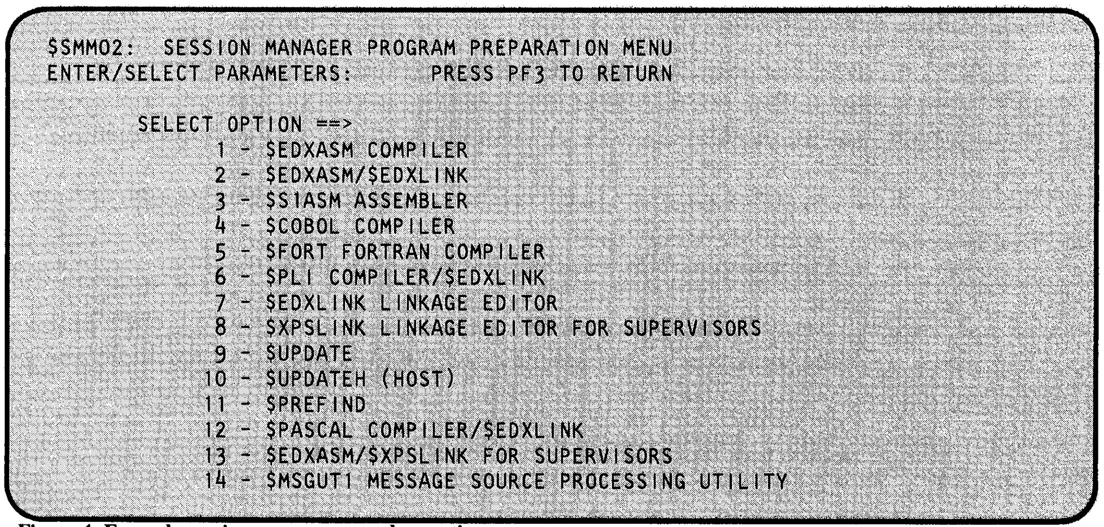
```
$SMMPRIM:  SESSION MANAGER PRIMARY OPTION MENU
ENTER/SELECT PARAMETERS:              PRESS PF3 TO EXIT
       SELECT OPTION ==>
         1 -  TEXT EDITING
         2 -  PROGRAM PREPARATION
         3 -  DATA MANAGEMENT
         4 -  TERMINAL UTILITIES
         5 -  GRAPHICS UTILITIES
         6 -  EXEC PROGRAM/UTILITY
         7 -  EXEC $JOBUTIL PROC
         8 -  COMMUNICATION UTILITIES
         9 -  DIAGNOSTIC AIDS
        10 -  BACKGROUND JOB CONTROL UTILITIES
```

Figure 3. Session manager primary option menu

Secondary Option Menu

A secondary option menu lists the utilities that are available under the related primary option. Primary options 2, 3, 4, 5, 8, 9, and 10 have secondary option menus. Figure 4 shows an example of the secondary option menu for primary option 2 - Program Preparation.

To select a secondary option, enter the number of the option on the SELECT OPTION prompt line. After you select a secondary option, the session manager either displays a parameter input menu or loads the requested utility.



```
$SMM02:  SESSION MANAGER PROGRAM PREPARATION MENU
ENTER/SELECT PARAMETERS:      PRESS PF3 TO RETURN

       SELECT OPTION ==>
             1 - $EDXASM COMPILER
             2 - $EDXASM/$EDXLINK
             3 - $S1ASM ASSEMBLER
             4 - $COBOL COMPILER
             5 - $FORT FORTRAN COMPILER
             6 - $PL1 COMPILER/$EDXLINK
             7 - $EDXLINK LINKAGE EDITOR
             8 - $XPSLINK LINKAGE EDITOR FOR SUPERVISORS
             9 - $UPDATE
            10 - $UPDATEH (HOST)
            11 - $PREFIND
            12 - $PASCAL COMPILER/$EDXLINK
            13 - $EDXASM/$XPSLINK FOR SUPERVISORS
            14 - $MSGUT1 MESSAGE SOURCE PROCESSING UTILITY
```

Figure 4. Example session manager secondary option menu

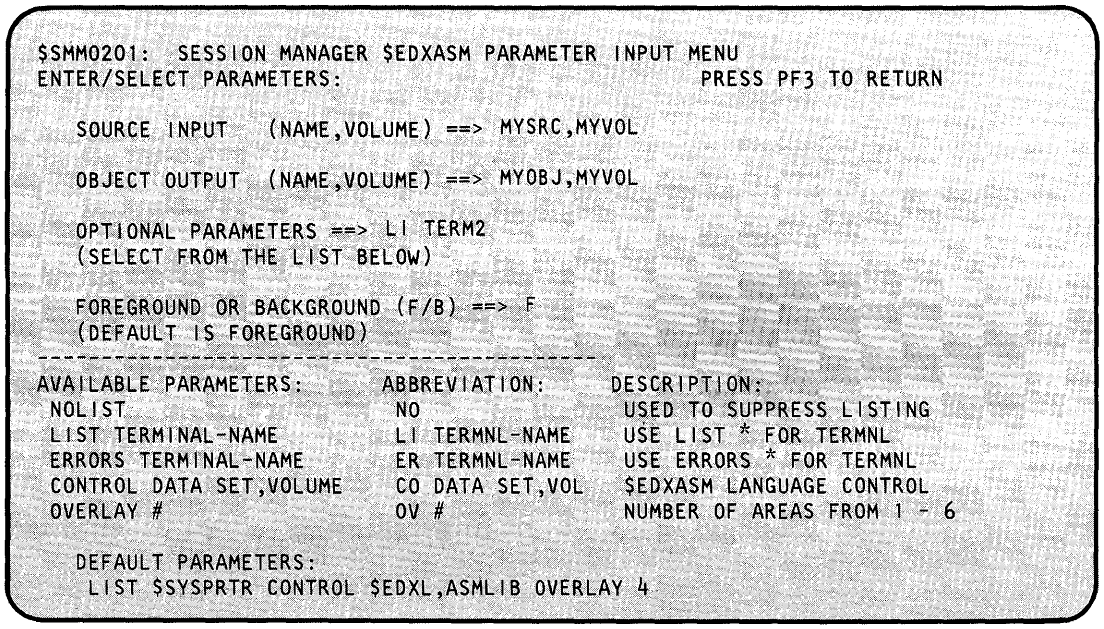
Menus *(continued)*

Parameter Input Menu

The parameter input menus prompt you for parameters, such as a data set and volume name, that are required by the requested utility.

Primary options 1, 6, and 7 have a parameter input menu but no secondary option menu. Figure 5 contains an example of the parameter input menu for the $EDXASM utility (primary option 2, secondary option 1).

Enter the requested parameters in the format expected by the requested utility.



```
$SMM0201:  SESSION MANAGER $EDXASM PARAMETER INPUT MENU
ENTER/SELECT PARAMETERS:                                  PRESS PF3 TO RETURN

   SOURCE INPUT   (NAME,VOLUME) ==> MYSRC,MYVOL

   OBJECT OUTPUT  (NAME,VOLUME) ==> MYOBJ,MYVOL

   OPTIONAL PARAMETERS ==> LI TERM2
   (SELECT FROM THE LIST BELOW)

   FOREGROUND OR BACKGROUND (F/B) ==> F
   (DEFAULT IS FOREGROUND)
--------------------------------------------
AVAILABLE PARAMETERS:       ABBREVIATION:     DESCRIPTION:
 NOLIST                      NO                USED TO SUPPRESS LISTING
 LIST TERMINAL-NAME          LI TERMNL-NAME    USE LIST * FOR TERMNL
 ERRORS TERMINAL-NAME        ER TERMNL-NAME    USE ERRORS * FOR TERMNL
 CONTROL DATA SET,VOLUME     CO DATA SET,VOL   $EDXASM LANGUAGE CONTROL
 OVERLAY #                   OV #              NUMBER OF AREAS FROM 1 - 6

   DEFAULT PARAMETERS:
    LIST $SYSPRTR CONTROL $EDXL,ASMLIB OVERLAY 4
```

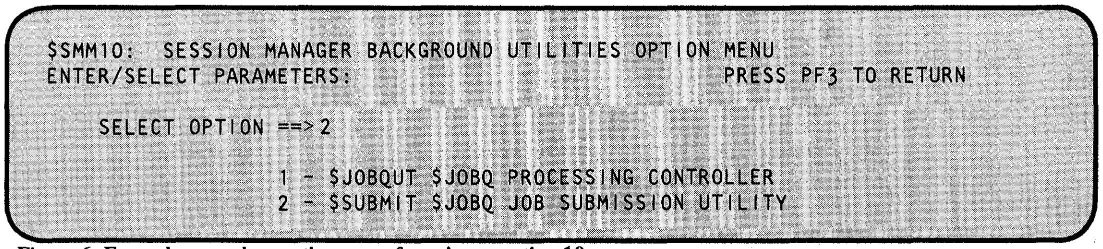
Figure 5. Example session manager parameter input menu

The Background Option

Figure 5 shows that you can specify either the background or foreground option. This choice is offered with options 2.1, 2.2, 2.7, 2.8, 2.13, 6, and 7. If you run a job in foreground, you cannot use your terminal until the job has completed. Foreground is adequate for small jobs. However, if you don't want to tie up your terminal waiting for a large job to complete, run in background and the job will run on another terminal. Then you can continue to use your terminal for another job.

If you try to code anything except an "F" or a blank for foreground or a "B" for background, you will receive an "INVALID PARAMETER INPUT" message.

To submit a job through the background options of 10.1 or 10.2, the system must load the batch control manager, $JOBQ. For a job to execute in background, 8K bytes of storage must be available.



```
$SMM10:  SESSION MANAGER BACKGROUND UTILITIES OPTION MENU
ENTER/SELECT PARAMETERS:                              PRESS PF3 TO RETURN

     SELECT OPTION ==> 2

                     1 - $JOBQUT $JOBQ PROCESSING CONTROLLER
                     2 - $SUBMIT $JOBQ JOB SUBMISSION UTILITY
```

Figure 6. Example secondary option menu for primary option 10

Note: See the $JOBQUT and $SUBMIT utilities for the screen examples.

The session manager allocates one additional work data set for the entire system to use for background processing. When you log onto the system, the session manager checks to see if this work data set exists already. If it doesn't, the session manager allocates 400 records for the data set. If the data set already exists, the session manager continues as usual. Every job submitted in background that needs a work data set will use this preallocated data set. Since only one job can run background at a time, there is no problem. If you delete this data set, the session manager will reallocate it when the next user logs on.

Note: If you never intend to run background jobs, your system manager can move this entry after the end statement in the data set $SMALLOC, EDX002 with $FSEDIT.

Custom Menus

You can add your own custom menus which give you access to your application programs with the session manager. Refer to the *Customization Guide* for instructions on customizing the session manager.

Data Sets

The session manager uses the following six work data sets for each person that is logged-on. (If your session manager has been tailored as described in the *Customization Guide*, additional data sets may be used.)

- $SMEuser
- $SMPuser
- $SMWuser
- $SM1user
- $SM2user
- $SM3user.

The data sets are allocated after you enter your user ID on the logon menu, unless they were saved at the end of a previous session. The session manager uses your user ID as part of the data set name, creating a unique set of data sets for each user.

Data Sets *(continued)*

When you log off, the session manager gives you an opportunity to erase all work data sets except $SMPuser. If you chose to save the data sets, the information they contain will be available the next time you sign on with the same user ID. Figure 7 lists the six basic session-manager data sets, their sizes, and their purpose.

Data set name	Size in 256-byte records	Purpose
$SMEuser	400	Used by the full-screen text editor ($FSEDIT) as a work data set.
$SMPuser	30	Used by the session manger to save your input parameters. This data set is not deleted at the end of a session, making your parameters available for the next session.
$SMWuser	30	Used by the session manger to submit procedures via the job procedure utility ($JOBUTIL).
$SM1user[1]	400	Used by $S1ASM, $EDXASM, $COBOL, $PL/I, $PASCAL and $FORT as a work data set.
$SM2user[1]	400	Used by $EDXLINK, $S1ASM, $EDXASM, $XPSLINK, $COBOL, $PL/I, and $FORT as a work data set.
$SM3user[1]	250	Used by the Series/1 Macro Assembler ($S1ASM), $COBOL, $PASCAL, and $PL/I as a work data set.

Figure 7. Session manager data sets

Note: If you use option 2.8 or 2.13, the session manager expands $SM2user to 600 records and then resets it to 400 records.

Program Function Keys

The session manager has four program function (PF) keys defined for special use: PF1, PF2, PF3, and PF4. They perform the following functions:

PF1 Suspends the session manager, allowing you to enter operator commands. Suspending the session manger is quicker than logging off and back on. To restart the session manager, press the attention key, and enter $SM. The session manager returns to the menu you were using when you pressed the PF1 key.

[1] These data sets must be deleted and reallocated to new sizes when using the session manger to invoke compilers and assemblers. Recommended sizes for most programs are 2000 records for $SM1USER and $SM2USER and 800 records for $SM3USER.

Program Function Keys *(continued)*

When you press the PF1 key, the session manager displays the following messages:

```
ENTERING SYSTEM COMMAND MODE -
TO REENTER THE SESSION MANAGER,
DEPRESS THE ATTN KEY AND ENTER "$SM"
```

PF2 Restores the current menu screen to its appearance when first displayed. Use this key to erase incorrect entries.

PF3 Returns to the previous menu.

PF4 Return directly to the primary option menu.

Supported Utilities

The following table lists the EDX system utilities that are supported by the session manager and the primary and secondary option numbers for each. (See Figure 1 on page UT-3 for a table which includes a list of the jobs that can be done with the session manager, including the appropriate primary and secondary option numbers for each.)

Note: The session manager menus are independent of the EDX supervisor installed on your EDX system. Therefore, all the utilities listed on the menu screens may not be a part of your system.

Utility	Description	Options
$ARJE	Advanced RJE program and utilities	8.10
$BSCTRCE	Trace I/O activity on a BSC line	8.1
$BSCUT1	Format BSC trace files	8.2
$BSCUT2	Communications I/O exerciser	8.3
$CHANUT1	Channel attach utility	8.9
$COBOL	COBOL language compiler	2.4
$COMPRES	Compress disk, diskette, or volume	3.4
$COPY	Copy data set or volume	3.5

Figure 8 (Part 1 of 3). Session Manager Options by Utility

Supported Utilities *(continued)*

Utility	Description	Options
$COPYUT1	Copy data set with allocation	3.3
$DASDI	Format disk or diskette	3.6
$DICOMP	Display and change graphics profiles	5.2
$DIINTR	Graphics interpreter	5.3
$DISKUT1	Allocate, delete, list data set directory data	3.1
$DISKUT2	Patch, dump, or list a data set or program a data set or program	3.2, 9.2
$DIUTIL	Maintain partitioned data base	5.1
$DUMP	Format and list saved environment	9.1
$EDXASM	Event Driven Language Compiler	2.1
$EDXASM/$EDXLINK	Compile and link	2.2
$EDXASM/$XPSLINK	Compile and link a supervisor program	2.13
$EDXLINK	Linkage editor	2.7
$FONT	Process 4974, 4978, and 4980 character image tables	4.5
$FORT	FORTRAN language compiler	2.5
$FSEDIT	Full-screen editor	1
$GPIBUT1	General purpose interface bus utility	4.9
$HCFUT1	Interact with Host Communications Facility	8.8
$HXUT1	H exchange diskette utility	3.11
$IAMUT1	Indexed Access Method Utility	3.9
$IMAGE	Define 4978, 4979, 4980 or 3101 screen image	4.4
$INITDSK	Initialize disk or diskette, and volume control	3.7
$IOTEST	Test sensor I/O; list hardware configuration	9.3
$JOBUTIL	Job stream processor	7

Figure 8 (Part 2 of 3). Session Manager Options by Utility

Supported Utilities *(continued)*

Utility	Description	Options
$JOBQUT	Job queue processing controller	10.1
$MOVEVOL	Disk volume dump and restore	3.8
$MSGUT1	Message-source processing	2.14
$PASCAL/$EDXLINK	Pascal language compile and link	2.12
$PFMAP	Display 4978 and 4980 PF key codes	4.6
$PLI/$EDXLINK	PLI language compile and link	2.6
$PREFIND	Prefind data sets and EDL overlays	2.11
$PRT2780	2780 spooled RJE file printer	8.6
$PRT3780	3780 spooled RJE file printer	8.7
$RJE2780	2780 remote job entry to host	8.4
$RJE3780	3780 remote job entry to host	8.5
$SPLUT1	Spool utility	4.7
$SUBMIT	Job queue job submission utility	10.2
$S1ASM	Series/1 assembler	2.3
$S1S1UT1	Series/1 to Series/1	4.8
$TAPEUT1	Tape management	3.10
$TERMUT1	Change terminal parameters	4.1
$TERMUT2	Change 4974, 4978, and 4980 image or control store	4.2
$TERMUT3	Send message to a terminal	4.3
$UPDATE	Converting Series/1 programs	2.9
$UPDATEH	Convert host system programs	2.10
$VERIFY	Verify Indexed Access Method Files	9.4
$XPSLINK	Link a supervisor program	2.8

Figure 8 (Part 3 of 3). Session Manager Options by Utility

Notes

Chapter 4. Utilities

The system utilities are a set of programs supplied with the Event Driven Executive. They allow you to interactively communicate with the system and perform tasks necessary for Series/1 application program development and system maintenance.

This chapter provides detailed descriptions (in alphabetical order) of the EDX system utility programs.

Invoking the Utilities

The Event Driven Executive provides three ways to invoke the utility programs from a terminal:

Session manager	You choose the desired utility program from a predefined option menu. This is the easiest to use for interactive utilities because you only enter option numbers (not program names) to access the function needed.
$JOBUTIL	The job stream processor utility is used to invoke a predefined sequence of utility programs and to pass parameters to those programs. $JOBUTIL can be invoked by the $L operator command or the session manager.
$L command	Enter the operator command $L (load program), followed by the utility name. All utilities described in this chapter can be invoked using $L.

Invoking the Utilities *(continued)*

Any utility invoked results in a loading message being displayed. The following example is for illustrative purposes only.

```
LOADING UTILITY  xP,00:00:00, LP= 0000, PART= number
```

Here, UTILITY is the name of the utility being loaded. xP indicates the size of the utility in pages (256 bytes equals one page). 00.00.00 is the time in hours, minutes, and seconds. LP= 0000 indicates that the load point of the utility is at location X'0000', and PART= number indicates the partition in which the utility is loaded. If timer support is not included in your supervisor, the time is not printed.

Most utility programs are used interactively from a terminal. If you are not familiar with the commands available under a specific utility, you can enter a question mark in response to the COMMAND (?): prompt and press the enter key. A list of the available commands for that utility is displayed.

Entering Utility Commands

You can enter utility commands in one of two ways — *prompt-reply* or *single-line* format. With prompt-reply format, you enter the command name and each parameter as the system asks for it. With single-line format, you enter the command name and all the parameters on the same line. The following examples show you how to use the two formats.

Prompt/Reply Format

Type the utility command following the COMMAND (?): prompt and press the enter key. EDX responds with a prompt for the next parameter as each parameter is entered.

```
COMMAND (?): CV

COPY VOLUME
ENTER SOURCE VOLUME: IBMEDX
ENTER TARGET VOLUME: EDX002
ENTER TARGET DATA SET NAME: DATA1
ARE ALL PARAMETERS CORRECT? Y
COPY COMPLETE
            949 RECORDS COPIED

COMMAND (?):
```

Invoking the Utilities *(continued)*

Single Reply Entry

Type the command name and all the required parameters (information) needed by the command to perform its function following the COMMAND (?): prompt and press the enter key. When using the single reply entry, the parameters must be entered in the order that the EDX expects them.

```
COMMAND (?):  CV IBMEDX EDX002 DATA1 Y
COPY COMPLETE
            949 RECORDS COPIED

COMMAND (?):
```

Cancelling a Utility

Use the $C operator command to cancel a utility program running in the same partition as your terminal. If $C should not be used to cancel a utility, the utility warns you on the first screen it displays.

To cancel a utility with the $C operator command:

1) Press the attention key.

2) EDX responds with the greater-than sign (>).

3) Enter the $C operator command.

4) Press the enter key.

$BSCTRCE - Trace I/O Activity on a BSC Line

The $BSCTRCE utility traces the I/O activities on a given binary synchronous communication (BSC) line. You must load $BSCTRCE in the same partition as the application program that is controlling the traced line. If you load it in any other partition, you will get unpredictable results.

Invoking $BSCTRCE

You invoke $BSCTRCE with the $L command or option 8.1 of the session manager.

After you load $BSCTRCE, it prompts you for the disk or diskette file in which to place the trace output. $BSCTRCE then prompts you for the line number you want traced. Use the attention command STOP to end the trace action.

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

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

When the system reaches the end of the output file, it reuses it from the beginning, since $BSCTRCE displays the relative record number of the last trace record it wrote before it ended. You can display or list the trace file by using the $BSCUT1 utility.

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

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

When $BSCTRCE ends, it displays the relative record number of the last trace record it wrote.

$BSCTRCE - Trace I/O Activity on a BSC Line *(continued)*

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

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

$BSCUT1 - Format BSC Trace Files

The $BSCUT1 utility formats BSC trace files (see $BSCTRCE utility) for printing to either $SYSPRTR or a terminal. You can select the record for the trace file to dump. The system will prompt you as necessary for information that the functions of $BSCUT1 require.

Invoking $BSCUT1

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

$BSCUT1 Commands

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

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

  USING VOLUME EDX002

  COMMAND(?): ?

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

  COMMAND (?):
```

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

Example: Figure 9 on page UT-49 shows invoking and using $BSCUT1 to display specified trace data records.

$BSCUT1 - Format BSC Trace Files *(continued)*

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

     DUMP OF TRACE FILE TRACE9 ON EDX002

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

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

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

8    DATA LENGTH =     2
9    1     1061

10   *****  RECORD  33  *****  CONTINUATION OF CHAINED OPERATION

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

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

     DUMP COMPLETE
     ANOTHER AREA?
```

Figure 9. Dumping BSC trace records to a terminal

1 The command for printing the selected records is DP; the command for dumping records to a terminal is DU. THIS EXAMPLE USES DU.

2 You can select a specific range of records for $BSCUT1 to display. Enter the first and last record numbers as the utility requests them. Figure 9 on page UT-46 shows that when $BSCTRCE ends, it displays the last trace record number. This information provides you with the range of record numbers that are available in the $BSCTRCE data set.

3 The system displays the record number along with the type of BSC operation it represents.

$BSCUT1 - Format BSC Trace Files *(continued)*

4 This line provides 3 items of diagnostic information:

CC	This is the interrupt condition code. In this example the 2 is the condition code returned and indicates an exception condition.
ISW	The interrupt status word represents the interrupt status byte in the left-most byte.
STATUS	The 3 words of status are the values stored in the cycle steal status words (CSSW) for the BSC adapter. The CSSWs contain valid diagnostic information when the ISW bit 0 is set to 1.

The detailed descriptions of these codes are contained in the *IBM Series/1 Communications Features Description*, GA34-0028.

When analyzing the trace records, you must remember that the system writes the trace records after the BSC operation completes. Therefore, the error indications may not relate directly to the record with which they are formatted and printed but will relate to the operation as a whole.

5 This line provides a brief interpretation of the condition code field and the interrupt status byte.

6 These 8 words represent the values stored in the device control block (DCB). Their meanings are described in detail in the *IBM Series/1 Communications Features Description*, GA34-0028.

7 This line provides a brief interpretation of what operation was performed by the device control block whose values are represented on the previous line.

8 This is the number of bytes of data that the current operation receives.

9 This is the first byte of data in the record the system is displaying. Notice that in record number 33 of the example, the DATA LENGTH = 485 (bytes). Also, the left-most column of the record's data that the system is displaying shows the first byte position of each line in decimal values. When the DATA LENGTH of a trace record exceeds 227 bytes, the system displays only the first 224 bytes of data followed by 'LAST 4' and the last 4 bytes of the data record.

10 This is the beginning of the display for LAST RECORD selected in the example, record number 33.

$BSCUT2 - Communications I/O Exerciser

The $BSCUT2 utility is primarily an I/O exerciser and is used to verify the following:

- Binary synchronous communications access method (BSCAM)
- BSCLINE definitions generated in the executing supervisor
- Customized jumper assignments in the BSC hardware features, such as:
  - device address
  - type of connection
  - tributary station address.

You can use $BSCTRCE to trace the exercising activities of $BSCUT2. You can format and print the records with $BSCUT1.

For each function you select in $BSCUT2, the system prompts you for the device (line) address, tributary station address (if multipoint), record length, and other related information. If any discrepancies exist between the function you are performing and the hardware assignments, the system prints error messages.

$BSCUT2 checks out binary synchronous operations if at least two binary synchronous adapters are available on Series/1 processors and if you make the connection between the two adapters. If you use a switched manual connection, $BSCUT2 does not prompt you to make the connection. This must be done after you issue the $BSCUT2 command and answer all prompts.

The BSCAM capabilities that $BSCUT2 can test are:

- Read and write of both transparent and nontransparent data
- Operation in limited conversational mode with both transparent and nontransparent data
- Operation as a control station on a multipoint line to both poll and select tributaries (text written only for transparent data)
- Operation as a tributary station on a multipoint line to be polled and selected (text written only for transparent mode)

Test Pattern Messages

$BSCUT2 issues a test pattern message for every record it read or wrote in a test.

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

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

$BSCUT2 - Communications I/O Exerciser *(continued)*

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

The meanings of the task names are as follows:

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

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

Invoking $BSCUT2

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

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

$BSCUT2 - Communications I/O Exerciser *(continued)*

$BSCUT2 Commands

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

```
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
```

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

Most of the commands and their explanations are presented in alphabetical order on the following pages.

CH — Change Hard-copy Device

Use the CH command to reassign the hard-copy device for the terminal or printer output.

Example:

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

Note: If the hard-copy device you specified is not defined, the system directs the output to the terminal where you loaded $BSCUT2.

EN — End $BSCUT2 Program

Use the EN command to end the $BSCUT2 utility.

Example:

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

$BSCUT2 - Communications I/O Exerciser *(continued)*

RI — Read Transparent/Nontransparent

The read task does not require NUMBER OF RECORDS since it will read either transparent or nontransparent data until the system receives EOT. This makes the read task useful for monitoring any BSC line sending data to the processor. For example, RI can receive data from the $RJE2780 or $RJE3780 utility operating in the same Series/1 or in another Series/1.

Note: The RI, WI, and WIX commands individually activate the tasks composing RWI and RWIX.

Example:

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

RWI — Read/Write Nontransparent Data

Use the RWI command to read and write nontransparent messages on a line. The system numbers each message. The record length for write includes the control characters. The read task receives the messages, analyzes them, and prints them on a hard-copy device. The analysis includes whether they are transparent or nontransparent and record length received.

Example:

```
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
```

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

$BSCUT2 - Communications I/O Exerciser *(continued)*

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

NUMBER OF RECORDS determines the number of transmissions the system will make before the test ends.

The MONITOR function causes each task to report its progress to the terminal. If you enable the monitor function, the system writes messages such as TASK ENTERED and TASK EXITED to the terminal.

RWIV — Read/Write Nontransparent Conversational

Use the RWIV to test limited conversational operation in nontransparent mode. BUFFER LENGTH is equivalent to RECL.

Example:

```
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
```

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

The RECL prompts refer to the buffer size the system will use and, therefore, the number of bytes the system transfers in one transmission over the BSC line. The maximum buffer size the system permits is 512 bytes. READ (RECL) should always be equal to or greater than WRITE (RECL) or errors will occur. NUMBER OF RECORDS determines the number of transmissions the system will make before the test ends. The MONITOR function causes each task to report its progress to the terminal. If the the system enables the monitor function, it writes messages such as TASK ENTERED and TASK EXITED to the terminal.

$BSCUT2 - Communications I/O Exerciser *(continued)*

The following is a description of the binary synchronous line transactions:

```
WRITE TASK                                                READ TASK

BSCWRITE IV(X)         ----ENQ------------->              BSCREAD I
                       <---ACK0 (Response)-
                       ----Text------------>
                       <---Text (Response)--              BSCWRITE CV(X)
BSCREAD C              ----ACK1 (Response)->
                       <---Text-------------              BSCWRITE CV(X)
BSCWRITE CV(X)         ----Text (Response)->
                       <---ACK0 (Response)--              BSCREAD C
BSCWRITE CV(X)         ----Text------------>
                       <---Text-------------              BSCWRITE CV(X)
BSCREAD C              ----ACK1------------>
```

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

RWIVX — Read/Write Transparent Conversational

Use the RWIVX command to test limited conversational operation in transparent mode. 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 a hard-copy device. The analysis includes whether they are transparent or nontransparent and record length received. BUFFER LENGTH is equivalent to RECL.

Example:

```
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
```

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

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

$BSCUT2 - Communications I/O Exerciser *(continued)*

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

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

The following is a description of the binary synchronous line transactions:

```
WRITE TASK                                                      READ TASK

BSCWRITE IV(X)              ----ENQ------------->               BSCREAD I
                            <---ACK0 (Response)-
                            ----Text------------>
                            <---Text (Response)--               BSCWRITE CV(X)
BSCREAD C                   ----ACK1 (Response)->
                            <---Text-------------               BSCWRITE CV(X)
BSCWRITE CV(X)              ----Text (Response)->
                            <---ACK0 (Response)--               BSCREAD C
BSCWRITE CV(X)              ----Text------------>
                            <---Text-------------               BSCWRITE CV(X)
BSCREAD C                   ----ACK1------------>
```

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

RWIX — Read/Write Transparent Data

Use the RWIX command to read and write transparent messages on a 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 a hard-copy device. The analysis includes whether they are transparent or nontransparent and record length received.

Example:

```
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
```

$BSCUT2 - Communications I/O Exerciser *(continued)*

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

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

NUMBER OF RECORDS determines the number of transmissions the system will make before the test ends.

The MONITOR function causes each task to report its progress to the terminal. If the system enables the monitor function, it writes messages such as TASK ENTERED and TASK EXITED to the terminal.

RWIXMP — Read/Write Transparent, Multidrop Line

Use the RWIXMP command to read and write transparent messages on a multidrop 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 a hard-copy device. The analysis includes whether they are transparent or nontransparent and record length received.

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

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

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

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

$BSCUT2 - Communications I/O Exerciser *(continued)*

In the following example, the control station (MC) at device address 50 polls and selects all tributary stations (MT) and sends and receives messages to them. Since each task both transmits and receives, successful operation requires the control station length to equal all tributary station buffer lengths. Values other than this can be entered to test access method error detection. Received messages are logged to the hard-copy device.

Example:

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

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

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

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

If this test is to be performed 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).
- NUMBER OF TRIBUTARIES must be equal in both programs.
- LOOP COUNT must be equal in both programs.

$BSCUT2 - Communications I/O Exerciser *(continued)*

WI — Write Nontransparent

Note: The RI, WI, and WIX commands individually activate the tasks composing RWI and RWIX.

Example:

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

WIX — Write Transparent

Note: The RI, WI, and WIX commands individually activate the tasks composing RWI and RWIX.

Example:

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

$CHANUT1 - Channel Attach Utility

The $CHANUT1 utility starts or stops a channel attach device, enables or disables I/O tracing, and prints the trace area. $CHANUT1 issues prompts to the terminal where you loaded it; in response to a prompt, you must enter a command.

Invoking $CHANUT1

Invoke $CHANUT1 with the $L command or option 8.9 of the session manager. When you load $CHANUT1, it prompts you for the address of the channel attach device.

```
 >  $L $CHANUT1
ENTER DEVICE ADDRESS (Hex):   10
COMMAND(?):
```

You can load the channel attach utility into any partition. The $CHANUT1 commands interface with the channel attach program in the same manner as the channel attach instructions. The error codes for the $CHANUT1 commands are the same as those for the corresponding instructions. See *Messages and Codes* for more information.

$CHANUT1 Commands

To display the $CHANUT1 commands at your terminal, enter a question mark in response to the prompting message COMMAND (?):.

```
COMMAND(?):    ?

    CA  --   CHANGE DEVICE ADDRESS
    EN  --   TERMINATE THE UTILITY
    PR  --   PRINT THE TRACE AREA
    ST  --   START A CHANNEL ATTACH DEVICE
    SP  --   STOP A CHANNEL ATTACH DEVICE
    TR  --   ENABLE/DISABLE TRACING
COMMAND(?):
```

After $CHANUT1 displays the commands, the system prompts you again with COMMAND (?):. Then you can respond with the command of your choice (for example, PR). Each command and its explanation is presented in alphabetical order on the following pages.

CA — Change Device Address

Use the CA command to change the device address.

```
COMMAND(?):   CA
ENTER DEVICE ADDRESS (HEX):   11
COMMAND(?):
```

$CHANUT1 - Channel Attach Utility *(continued)*

EN — End $CHANUT1 utility

Use the EN command to end the $CHANUT1 utility.

```
COMMAND(?):   EN
$CHANUT1 ENDED AT 1:14:40
```

PR — Print the Trace Area

Use the PR command to print the trace buffer, with the title you enter, on a terminal.

```
COMMAND(?):   PR
TITLE:  TRACE PRINTOUT 07/26/80
CONSOLE:  $SYSPRTR
PRINT TRACE BUFFER SUCCESSFUL
COMMAND(?):
```

SP — Stop a Channel Attach Device

Use the SP command to stop the channel attach device you have specified.

```
COMMAND(?):   SP
STOP DEVICE SUCCESSFUL
COMMAND(?):
```

ST — Start a Channel Attach Device

Use the ST command to start the channel attach device you have selected.

```
COMMAND(?):   ST
START DEVICE SUCCESSFUL
COMMAND(?):
```

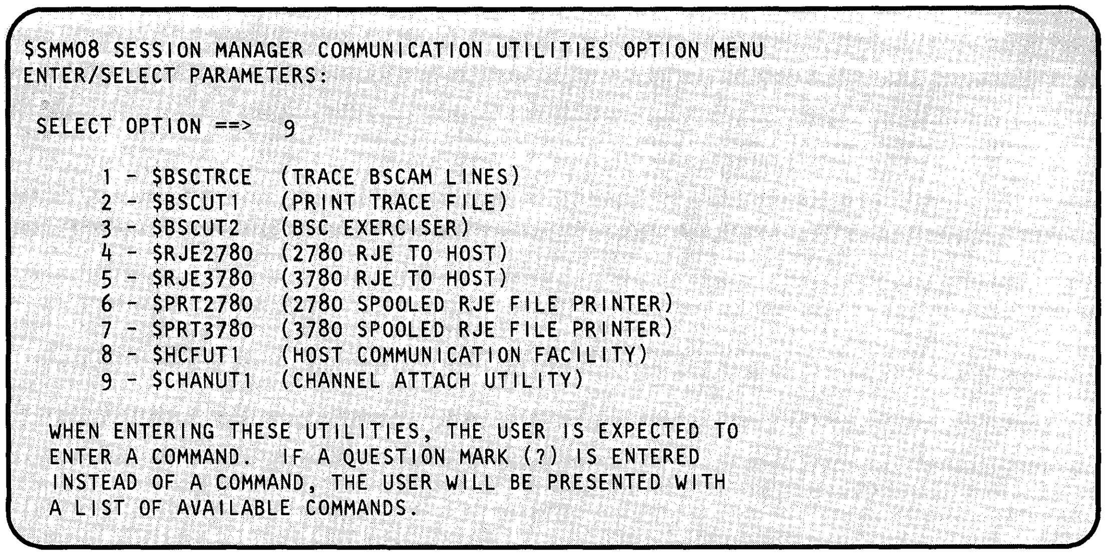
TR — Enable/Disable Trace

Use the TR command to enable (E) or disable (D) the trace function.

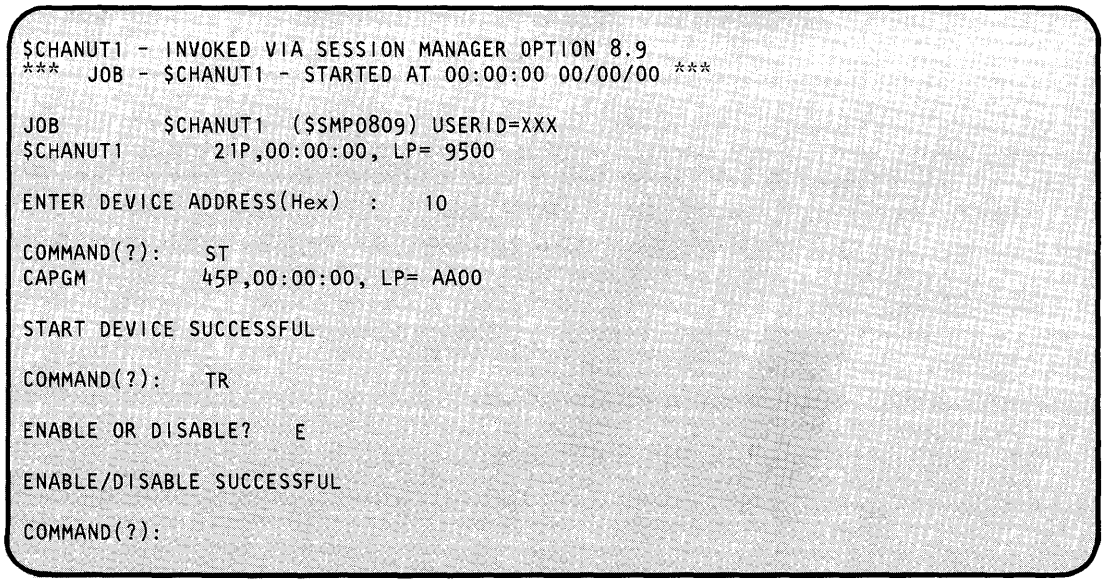
```
COMMAND(?):   TR
ENABLE OR DISABLE:   D
ENABLE/DISABLE SUCCESSFUL
COMMAND(?):
```

$CHANUT1 - Channel Attach Utility *(continued)*

The following is an example of starting a trace using the session manager.


```
$SMM08 SESSION MANAGER COMMUNICATION UTILITIES OPTION MENU
ENTER/SELECT PARAMETERS:

 SELECT OPTION ==>  9

      1 - $BSCTRCE  (TRACE BSCAM LINES)
      2 - $BSCUT1   (PRINT TRACE FILE)
      3 - $BSCUT2   (BSC EXERCISER)
      4 - $RJE2780  (2780 RJE TO HOST)
      5 - $RJE3780  (3780 RJE TO HOST)
      6 - $PRT2780  (2780 SPOOLED RJE FILE PRINTER)
      7 - $PRT3780  (3780 SPOOLED RJE FILE PRINTER)
      8 - $HCFUT1   (HOST COMMUNICATION FACILITY)
      9 - $CHANUT1  (CHANNEL ATTACH UTILITY)

  WHEN ENTERING THESE UTILITIES, THE USER IS EXPECTED TO
  ENTER A COMMAND.  IF A QUESTION MARK (?) IS ENTERED
  INSTEAD OF A COMMAND, THE USER WILL BE PRESENTED WITH
  A LIST OF AVAILABLE COMMANDS.
```

The following shows what happens when you select option 9 (from the previous screen), request and start device 10, and enable trace.


```
$CHANUT1 - INVOKED VIA SESSION MANAGER OPTION 8.9
***  JOB - $CHANUT1 - STARTED AT 00:00:00 00/00/00 ***

JOB        $CHANUT1  ($SMP0809) USERID=XXX
$CHANUT1       21P,00:00:00, LP= 9500

ENTER DEVICE ADDRESS(Hex)  :   10

COMMAND(?):   ST
CAPGM         45P,00:00:00, LP= AA00

START DEVICE SUCCESSFUL

COMMAND(?):   TR

ENABLE OR DISABLE?   E

ENABLE/DISABLE SUCCESSFUL

COMMAND(?):
```

$COMPRES - Compress Disk, Diskette, or Volume

$COMPRES compresses a disk/diskette volume or the entire contents of a device. Used it to allocate new data sets and volumes when a volume or device is fragmented (due to deletion of data sets and volumes).

Notes:

1. Do not compress a volume or device while it is being accessed. Use the $A ALL command to determine if programs are active in the partition you are currently assigned to or in other partitions. However, if you are executing under the session manager, the $SMUuser program will be in the system but not active until $COMPRES ends.

2. You must initialize the IPL text after using $COMPRES if the device or the IPL volume you are compressing contains the supervisor ($EDXNUCX) and if the nucleus has moved.

3. Before compressing the IPL volume, you should create an IPLable diskette as follows:

   a. Use $DASDI to format a diskette.

   b. Use $INITDSK (ID command) to initialize the diskette. Do not allocate a nucleus if your customized nucleus EDXNUCX is larger than 400 records.

   c. Copy (using $COPYUT1) $EDXNUCX, $LOADER and $INITDSK to a backup IPL diskette. If a 4978 or 4980 terminal is your $SYSLOG device, you need to do a "copy generic" (CG) for $4978 and $4980. You will need to use the "copy member" command (CM) of $MFARAM, $FPCARAM, or $ACCARAM if you are using a 3101 or an ACCA device as $SYSLOG.

   d. Use the $INITDSK II command to initialize IPL text on the backup IPL diskette.

   e. Verify that you can IPL the diskette and load $INITDSK to initialize IPL text on the IPL volume on disk before starting the $COMPRES.

4. If the compress moves the nucleus (as indicated by the message $EDXNUCX COPIED):

   a. IPL from the backup diskette.

   b. Initialize (write IPL text) the nucleus on disk using the II command of $INITDSK.

   c. IPL from the disk.

5. If you have not copied $EDXNUCX, $LOADER, and $INITDSK to a backup diskette and the compress does move the nucleus, you may use the starter system to load $INITDSK (II command) to initialize the IPL text. This assumes that one of the attached terminals will be recognized by the starter system as $SYSLOG.

6. Compressing a volume may relocate EDL overlay programs and data sets which you have defined previously to $PREFIND. If this is the case, you must reissue $PREFIND.

$COMPRES - Compress Disk, Diskette, or Volume *(continued)*

7. Compressing a disk may relocate volumes which you defined as "performance volumes." If this is the case, you must re-IPL.

Specifying Dynamic Storage

To increase program performance you can change the dynamic storage used by $COMPRES. $COMPRES is shipped with a dynamic storage of 512 bytes. Using the $L command, you can specify the number of bytes of additional storage the system should allocate when $COMPRES is loaded for execution.

The following example shows how to temporarily change a 512 byte Dynamic Storage Allocation to 10K.

```
 > $L $COMPRES,,10240
```

You can use $DISKUT2 to modify the default load time storage allocation. See "SS — Set Program Storage Parameter" on page UT-219 for an example.

Invoking $COMPRES

Invoke $COMPRES with the $L command or option 3.4 of the session manager.

```
 > $L $COMPRES
LOADING $COMPRES     37P,08:00:00, LP = 5B00

$COMPRES - COMPRESS UTILITY

VOLUME/DEVICE COMPRESS
WARNING!  SHOULD BE RUN ONLY WHEN
NO OTHER PROGRAMS ARE ACTIVE

COMMAND (?):   ?

ROLLON     - SET SCREEN => NO PAUSE
ROLLOFF    - RESTORE STARTING CHARACTERISTICS
D          - DEVICE COMPRESS
V          - VOLUME COMPRESS
HF         - ESTIMATE COMPRESS PROGRESS
EN         - END

COMMAND (?):
```

$COMPRES Commands

Each command and its explanation is presented in alphabetical order on the following pages.

$COMPRES - Compress Disk, Diskette, or Volume *(continued)*

? — Help

Use the ? command if you want to see the command menu again.

Example: Help command.

```
COMMAND (?):    ?


ROLLON     - SET SCREEN => NO PAUSE
ROLLOFF    - RESTORE STARTING CHARACTERISTICS
D          - DEVICE COMPRESS
V          - VOLUME COMPRESS
<ATTN> HF  - ESTIMATE COMPRESS PROGRESS
EN         - END

COMMAND (?):
```

$COMPRES - Compress Disk, Diskette, or Volume *(continued)*

D — Device Compress

Use the D command to compress the entire contents of a device. Use $A ALL to determine if any programs are active before you compress the library.

Example: Compress a Device with Fixed-Head Volumes.

```
(Use $A ALL to determine if any programs are active
before you compress the volume)

COMMAND (?):   D
DEVICE ADDRESS:   48

COMPRESS DEVICE AT ADDRESS 0048 (Y/N)?   Y

DIRECTORY HAS BEEN SORTED BY MEMBER IN ASCENDING ORDER.

THE FIXED-HEAD VOLUME ==> FIXVOL NOT COPIED
EDX002   COPIED
EDITWORK COPIED
SRCLIB   COPIED
OBJLIB   COPIED
ASMLIB   COPIED
$COMPRES COPIED
THE DEVICE IS COMPRESSED.

ANOTHER COMPRESS?  (Y/N)?  N
```

EN — End $COMPRES

Use the EN command to end the $COMPRES utility.

Example: End $COMPRES utility

```
COMMAND (?):   EN

$COMPRES ENDED AT 08:03:59
```

$COMPRES - Compress Disk, Diskette, or Volume *(continued)*

HF — Estimate Compress Progress

Use >HF during a volume or device compress to get the percent of completion, the total number of records to be compressed, and the number already copied up to that point. To use this command, press the attention key and type in "HF".

Example 1: Estimate progress of volume directory compress.

```
  > HF

NOW AT:  70 PERCENT OF DIRECTORY LEFT TO BE SORTED
TOTAL NUMBER OF RECORDS:       162
NUMBER COPIED SO FAR   :        49


DIRECTORY SORTED
DIRECTORY HAS BEEN SORTED BY LOCATION IN ASCENDING ORDER.
$ASM000P ALREADY IN PLACE AND NOT COPIED
$EDXATSR ALREADY IN PLACE AND NOT COPIED
EDITINC  ALREADY IN PLACE AND NOT COPIED
                .
                .
                .
CDRRCB   COPIED
HEADER   COPIED
TDBEQU   COPIED
```

Example 2: Estimate progress of volume compress.

```
  > HF

COMPRESSING THE VOLUME/DISK, NOW AT:  15 PERCENT

$SPM     COPIED
COPCHECK COPIED
$ASMEXIO COPIED
  > HF
$ASM000H COPIED

COMPRESSING THE VOLUME/DISK, NOW AT:  17 PERCENT
CDRRCB   COPIED
HEADER   COPIED
TDBEQU   COPIED
        .
        .
        .
```

$COMPRES - Compress Disk, Diskette, or Volume *(continued)*

ROLLOFF — Restore Starting Characteristics

Use the ROLLOFF command to restore a terminal to its original mode. With ROLLOFF you must press enter each time the screen fills up.

Example: Restore starting characteristics.

```
COMMAND (?): ROLLOFF

COMMAND (?):
```

ROLLON — Set Screen => No Pause

Use the ROLLON command if you don't want to have to press the enter key each time the screen fills up. When you specify this command, the output "rolls" off the top of the screen as new terminal output appears at the bottom of the screen.

Example: Set screen to roll mode.

```
COMMAND (?): ROLLON

COMMAND (?):
```

$COMPRES - Compress Disk, Diskette, or Volume *(continued)*

V — Volume Compress

Use the V command to compress a disk or diskette volume. Use $A ALL to determine if any programs are active before you compress the library.

Example 1: Compress a Volume.

```
 > $A ALL

PROGRAMS AT 08:14:19
IN PARTITION #1 (STATIC) NONE
            PART.ADDR: 0000 HEX; SIZE: 12288 DEC.BYTES

PROGRAMS AT 08:14:19
IN PARTITION #2 (DYNAMIC)
$RMU       4700   TERMINAL 22
   CDROV2     5200
$DISKUT2   6F00   TERMINAL 15
            PART.ADDR: 4700 HEX; SIZE: 45056 DEC.BYTES

PROGRAMS AT 08:14:19
IN PARTITION #3 (STATIC) NONE
            PART.ADDR: 0000 HEX; SIZE: 45056 DEC.BYTES
            .
            .
            .
PROGRAMS AT 08:14:19
IN PARTITION #16 (DYNAMIC) NONE
            PART.ADDR: 0000 HEX; SIZE: 45056 DEC.BYTES

 > $L $COMPRES
LOADING $COMPRES    37P,08.15.35, LP=4C00

$COMPRES - COMPRESS UTILITY

VOLUME/DEVICE COMPRESS
WARNING!  SHOULD BE RUN ONLY WHEN
NO OTHER PROGRAMS ARE ACTIVE

COMMAND (?):   V
VOLUME LABEL = EDX001

COMPRESS VOLUME EDX001 (Y/N)?   Y

DIRECTORY HAS BEEN SORTED BY MEMBER IN ASCENDING ORDER.

MYPROG   COPIED
PROG1    COPIED
PROG2    COPIED
PAYROLL  COPIED
THE VOLUME IS COMPRESSED.

ANOTHER COMPRESS (Y/N)?   N
WANT TO SORT VOLUME DIRECTORY (Y/N)? N
$COMPRES ENDED AT 08.16.03
```

$COMPRES - Compress Disk, Diskette, or Volume *(continued)*

Example 2: Compress the IPL volume.

```
(Use $A ALL to determine if any programs are
active before you compress the library)

> $L $COMPRES
LOADING $COMPRES     37P,08:16:20, LP = CA00

$COMPRES - COMPRESS UTILITY

VOLUME/DEVICE COMPRESS
WARNING!  SHOULD BE RUN ONLY WHEN
NO OTHER PROGRAMS ARE ACTIVE

COMMAND (?):  V
VOLUME LABEL = EDX002

COMPRESS VOLUME EDX002 (Y/N)?  Y

CAUTION:  YOU ARE ABOUT TO COMPRES THE IPL VOLUME.
IF THE NUCLEUS IS MOVED, THEN THIS VOLUME WILL NOT IPL
AGAIN UNTIL THE "II" COMMAND OF "$INITDSK" IS REISSUED.
ALSO IF "$LOADER" IS MOVED, NO PROGRAMS CAN BE LOADED.
DO YOU WISH TO PROCEED(Y/N)?  Y

DIRECTORY HAS BEEN SORTED BY MEMBER IN ASCENDING ORDER.

$EDXNUC  ALREADY IN PLACE AND NOT COPIED
$LOADER  COPIED
$TERMUT1 COPIED
$COPYUT1 COPIED
D3       COPIED
D4       COPIED
D5       COPIED
D6       COPIED
D7       COPIED
D8       COPIED
D9       COPIED
D0       COPIED
$COMPRES COPIED
THE VOLUME IS COMPRESSED.

ANOTHER COMPRESS (Y/N)? N
WANT TO SORT VOLUME DIRECTORY (Y/N)? N
$COMPRES ENDED AT 08.16.03
```

$COPY - Copy Data Set

$COPY copies a disk or diskette data set, in part or in its entirety, to another disk or diskette data set.

Copying Programs or Data Members

When copying volume members, the target member must already exist (allocate using $DISKUT1) and must be of the same organization as the source member. Two types of organization are available:

DATA Data sets used as work files, user source modules, and application data set.

When you copy data members, you may copy an entire member or only a selected number of records (partial copy). If you are copying the entire member, the target data member must be equal to or larger than the source. If you are doing a partial copy, the target member need not be as large as the source but must have enough space following the starting target record number to accommodate the number of records you are copying from the source member.

PROGRAM Data sets that will contain executable (loadable) Event Driven Executive programs.

When you copy program members, the target member must be equal to or greater than the source member.

Copying Disk/Diskette Volumes to Another Diskette/Disk

When you copy a single volume diskette to disk, the target data set size must be equal to or greater than the diskette size in records. When you copy a disk volume or a multivolume diskette volume to another disk volume or a multivolume diskette, both volumes should be equal in size. If the source volume is larger than the target, you are prompted for the name of the source data set you wish copied to the target. The system copies the source data set to the target volume starting at the absolute beginning ($$EDXVOL). If the source volume is smaller than the target, you are prompted for the name of the target data set into which you want the source volume copied.

Note: For information on copying H-exchange volumes see "$HXUT1 - H-Exchange Utility" on page UT-371.

Absolute Record Copy

$COPY provides an absolute record capability using the special system data set names $$, $$EDXLIB, and $$EDXVOL. This allows you to copy a record relative to the beginning of a device ($$EDXVOL) or relative to the beginning of a volume ($$EDXLIB). You can use this capability when you copy one single-volume diskette volume to another. The CV command of $COPY does not copy the first cylinder on diskette. If the source diskette were an IPL volume (has IPL text and $EDXNUC), the system would not copy the IPL text, contained in the first record of the first

$COPY - Copy Data Set *(continued)*

cylinder, to the target diskette. Therefore, the target diskette volume, although containing a supervisor in $EDXNUC, would not be able to load that supervisor when you pressed the IPL key.

To copy the IPL text to the target diskette, use the CD command with $$EDXVOL specified as the data set name and record 1 specified as the first and last record you want copied.

Note: $$, $$EDXVOL, and $$EDXLIB are special system data set names and you must use them with care. $$ is a reserved system name, $$EDXVOL points to the beginning of the device volume, and $$EDXLIB points to the beginning of the data set directory within a volume.

Specifying Dynamic Storage

To increase program performance you can change the dynamic storage used by $COPY. $COPY is shipped with a dynamic storage of 2K. Using the $L command, you can specify the number of bytes of additional storage the system should allocate when $COPY is loaded for execution.

The following example shows how to temporarily change a 2K Dynamic Storage Allocation to 20K.

```
> $L $COPY,,20480
```

$DISKUT2.can also modify the default load time storage allocation associated with a program using the SS command. See "$DISKUT2 - Patch/Dump/List Data Set or Program" on page UT-198 for an example.

Invoking $COPY

You invoke $COPY with the $L operator command or option 3.5 of the session manager.

```
> $L $COPY
LOADING $COPY    40P,00:12:04, LP= 9200, PART=1

$COPY - COPY UTILITY

COMMAND (?):  ?
```

$COPY - Copy Data Set *(continued)*

$COPY Commands

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

```
COMMAND (?):  ?

CD - COPY DATA SET
CV - COPY VOLUME
RE - COPY FROM BASIC EXCHANGE
WE - COPY TO BASIC EXCHANGE
 (-CA- WILL CANCEL)
EN - END PROGRAM
COMMAND (?):
```

After $COPY displays the commands, it prompts you again with COMMAND (?):. Then you can respond with the command of your choice (for example, CV). Each command and its explanation is presented in alphabetical order on the following pages.

CD — Copy Data Set

Use the CD command to copy disk or diskette data sets to a preallocated disk or diskette target data set. When you copy data sets, you may copy an entire data set or only a select number of records. If you are copying the entire data, the target data set must be equal to or larger than the source. If you do a partial copy, the target data set need not be as large as the source but should have enough space following the starting record number to accommodate the number of records you are copying from the source data set.

$COPY - Copy Data Set *(continued)*

Example 1: Copy entire data set.

```
COMMAND (?): CD

SOURCE (NAME,VOLUME): DATAFIL1
COPY ENTIRE DATA SET? Y
TARGET (NAME,VOLUME): DATAFIL2,EDX002
ARE ALL PARAMETERS CORRECT? Y
COPY COMPLETE
          50 RECORDS COPIED

COMMAND (?):
```

Notes:

1. You cannot copy data sets allocated as program organization to a data set that you allocated as data organization.

2. When you copy program members, the target and source data sets must be equal in size.

Example 2: Partial copy of a data set.

```
COMMAND (?): CD

SOURCE (NAME,VOLUME): DATAFIL1
COPY ENTIRE DATA SET? N
FIRST RECORD: 1
LAST RECORD: 3
TARGET (NAME,VOLUME): DATAFIL2
FIRST RECORD: 1
ARE ALL PARAMETERS CORRECT? Y
COPY COMPLETE
            3 RECORDS COPIED

COMMAND (?):
```

If the target data set is too small to accommodate the amount of data you are copying from the source data set, the utility issues following message:

```
            *** TARGET DATA SET TOO SMALL ***
            *** COPY REQUEST CANCELLED ***
```

When the output data set is on disk or diskette, the system updates the end-of-data pointers.

$COPY - Copy Data Set *(continued)*

CV — Copy Volume

Use the CV command to copy entire volumes. This provides a volume backup capability. You may copy a disk volume to a disk or diskette volume, a diskette volume to a diskette volume, or a diskette volume to a preallocated disk data set of appropriate size in records. The number of records for the various types of diskettes are:

Type	128 Bytes/ sector	256 Bytes/ sector	1024 Bytes/ sector
Diskette 1	949	1110	-
Diskette 2	1924	2200	-
Diskette 2 D	-	3848	4736

Volume copy operations do not add the members in a source volume to the target volume. The system replaces the original contents of the target volume, including the directory.

If you have two or more diskette units, you may perform diskette volume copies between diskette devices. If you have a single diskette drive and a disk, you can perform copies using the following procedure:

1. Allocate a target data set on a disk of appropriate size.

2. Using the CV command, copy the diskette volume to the disk data set.

3. Mount the target diskette on the diskette device and vary the device online.

4. Using the CV command, copy the contents of the disk data set to the target diskette.

If you have a single 4966 Diskette Magazine Unit and a disk, the above procedure is also recommended.

Example: Copy a diskette to a backup data set on a 4962 disk.

```
COMMAND (?):  CV

COPY VOLUME
ENTER SOURCE VOLUME:   IBMEDX
ENTER TARGET VOLUME:  EDX002
ENTER TARGET DATA SET NAME:   DATA1
ARE ALL PARAMETERS CORRECT?  Y
COPY COMPLETE
            949 RECORDS COPIED

COMMAND (?):
```

$COPY - Copy Data Set *(continued)*

The CV command copies the entire single volume diskette volume. Therefore, the target data set should be equal to or greater than the volume size in records. If the target data set is not large enough, you may choose to do a partial copy or allocate (using $DISKUT1) a target data set large enough to accommodate the source.

Note: You can perform CV on an entire multivolume diskette.

If the target data set is not large enough, you are prompted as follows:

```
SOURCE DATA SET HAS nn RECORDS
TARGET VOLUME HAS ONLY nn
DO YOU WISH TO CONTINUE? (Y/N):
```

If you respond Y, the system copies the source to the target data set until the target is full. If you respond N, the CV command ends and you are prompted for another command, COMMAND (?):.

Note: Once you have copied a volume to a target disk volume, the system replaces the original contents of the target volume, including the directory. As a result, you can no longer access the original contents of the target disk volume.

EN — End $COPY Utility

Use the EN command to end the $COPY utility.

```
COMMAND (?):  EN
$COPY ENDED AT 1:14:40
```

$COPY - Copy Data Set *(continued)*

RE — Copy from Basic Exchange

Use the RE command to copy a basic-exchange data set from a diskette to a disk data set. A basic-exchange data set is contained on a diskette that you formatted for Standard for Information Interchange. You can use only one-sided, 128-byte diskettes as EDX recognizes only one volume on a basic-exchange diskette. You must allocate the target disk data set using $DISKUT1 before you use the RE command.

RE prompts you for the source diskette data set name and volume, the target disk data set name and volume, the number of the first record you want written to the target data set, and the basic-exchange data set name.

Example 1: Copy entire basic-exchange diskette data set to disk.

```
COMMAND (?):  RE

SOURCE ($$EDXVOL,VOLUME):  $$EDXVOL,IBMEDX
TARGET (NAME,VOLUME):   DATAFIL1,EDX002

SPECIFY START/END? (Y/N):  N

ENTER BASIC-EXCHANGE DATA SET NAME:  DATA
NUMBER OF RECORDS COPIED:  52
COPY COMPLETED

COMMAND (?):
```

Note: If you enter the wrong data set name, the system issues a read/write error message.

Example 2: Copy basic-exchange data set to disk. The record number where the copy is to start on target disk is specified.

```
COMMAND (?):  RE

SOURCE ($$EDXVOL,VOLUME):  $$EDXVOL,IBMEDX
TARGET (NAME,VOLUME):  DATAFIL1,EDX002

SPECIFY START/END? Y/N:  Y
FIRST RECORD:  10

ENTER BASIC-EXCHANGE DATA SET NAME:  DATA
NUMBER OF RECORDS COPIED:  151
COPY COMPLETED

COMMAND (?):
```

$COPY - Copy Data Set *(continued)*

WE — Copy to Basic Exchange

Use the WE command to copy a disk data set to a basic-exchange data set on diskette. You must allocate the diskette data set before you use the WE command. Use $DASDI to format the diskette for Standard for Information Interchange. Under this format, $DASDI formats a volume called IBMEDX, initializes the basic-exchange header on the diskette, and automatically allocates a data set named DATA. DATA consists of all the data tracks on the diskette.

WE prompts you for the source disk data set name and volume, the starting or ending records, the target diskette data set name and volume, and the basic-exchange data set name.

Example 1: Copy a disk data set to a basic-exchange diskette.

```
COMMAND (?):  WE

SOURCE (NAME,VOLUME):  DATAFIL1,EDX002

SPECIFY START/END? (Y/N):  N
TARGET ($$EDXVOL,VOLUME):  $$EDXVOL,IBMEDX

ENTER BASIC EXCHANGE DATA SET NAME:  DATA
COPY COMPLETE

COMMAND (?):
```

Example 2: Copy a disk data set to a basic-exchange diskette. The beginning and ending record numbers on the disk to be copied to the target diskette are specified.

```
COMMAND (?):  WE

SOURCE (NAME,VOLUME):  DATAFIL1,EDX002

SPECIFY START/END? (Y/N):  Y
FIRST RECORD:  100
LAST RECORD:  150
TARGET ($$EDXVOL,VOLUME):  $$EDXVOL,IBMEDX

ENTER BASIC-EXCHANGE DATA SET NAME:  DATA
COPY COMPLETE

COMMAND (?):
```

Notes:

1. Errors may occur if you have not initialized the diskette. The system reads and writes data on the diskette two sectors per I/O operation.

2. The diskette data set you access must start on an odd sector boundary.

$COPYUT1 - Copy Data Set with Allocation

$COPYUT1 performs several related copy functions. These functions determine the size and organization of the source data set(s) that $COPYUT1 copies, allocate members on the target volume, and then copy the source member(s) to the target member(s). With $COPYUT1, you can copy one member using the CM command or you can use the multiple copy commands to copy all members from source to target volumes.

Notes:

1. Do not specify the dynamic storage option (for example, $L $COPYUT1,,48000) when loading $COPYUT1 to make copies of the same volume from two or more terminals at the same time. This will cause formatting errors in your table of contents.
2. If a member already exists on the target volume, it is first deleted, then reallocated when the new source is copied to the target volume. This occurs only if enough contiguous space is available for reallocation of the member. There are no prompting messages asking if you wish to replace the existing member.

For any copying related to tape, see "$TAPEUT1 - Tape Management" on page UT-522.

Specifying Dynamic Storage

To increase program performance you can change the dynamic storage used by $COPYUT1. $COPYUT1 is shipped with a dynamic storage of 2K. Using the $L command, you can specify the number of bytes of additional storage the system should allocate when $COPYUT1 is loaded for execution.

The following example shows how to temporarily change a 2K Dynamic Storage Allocation to 20K.

```
> $L $COPYUT1,,20480
```

You can use $DISKUT2. to modify the default load time storage allocation See "SS — Set Program Storage Parameter" on page UT-219 for an example.

Invoking $COPYUT1

Invoke $COPYUT1 with the $L command or option 3.3 of the session manager.

$COPYUT1 - Copy Data Set with Allocation *(continued)*

```
 > $L $COPYUT1
LOADING $COPYUT1   59P,00:01:54, LP= 0000, PART= 2

$COPYUT1 - DATA SET COPY UTILITY

         *** WARNING ***
MEMBERS ON TARGET VOLUME WILL BE DELETED
REALLOCATION AND COPYING OF MEMBERS IS
DEPENDENT ON SUFFICIENT CONTIGUOUS SPACE

THE DEFINED SOURCE VOLUME IS EDX002  OK (Y/N)?  Y
THE DEFINED TARGET VOLUME IS EDX002  OK (Y/N)?  Y
MEMBER WILL BE COPIED FROM EDX002 TO EDX002  OK (Y/N)?  Y

COMMAND (?):
```

When you invoke $COPYUT1, the source and target volumes are assumed to be on the IPL volume. You have the option of specifying the source and target volumes. Once you specify the correct volumes, the commands copy members from the source volume to the target volume until you change the volume using the CV command.

$COPYUT1 Commands

To display the $COPYUT1 commands at your terminal, enter a question mark in response to the prompting message COMMAND (?):.

```
COMMAND (?): ?

====== MODE COMMANDS ======
WV       -- VERIFY COPY USING WRITE VERIFY
RDBK     -- VERIFY COPY USING READBACK CHECK
SQ       -- SET PROMPT MODE FOR COPY COMMANDS
NQ       -- RESET PROMPT MODE FOR COPY COMMANDS
CV       -- CHANGE SOURCE AND TARGET VOLUMES
ROLLON   -- SET SCREEN => NO PAUSE
ROLLOFF  -- RESTORE PAUSE CHARACTERISTICS
====== COPY COMMANDS ======
CM       -- COPY MEMBER FROM SOURCE TO TARGET
CALL     -- COPY ALL MEMBERS FROM SOURCE TO TARGET
CAD      -- COPY ALL DATA MEMBERS FROM SOURCE TO TARGET
CAP      -- COPY ALL PROGRAMS FROM SOURCE TO TARGET
CG       -- COPY ALL MEMBERS STARTING WITH TEXT FROM ...
CNG      -- COPY ALL MEMBERS NOT STARTING WITH TEXT FROM ...
===========================
<ATTN> CA -- CANCEL MULTIPLE COPY COMMANDS
EN       -- END PROGRAM
?        -- HELP

COMMAND (?):
```

After $COPYUT1 displays the commands, it prompts you with COMMAND (?): again. Then you can respond with the command of your choice (for example, CM).

$COPYUT1 - Copy Data Set with Allocation *(continued)*

The Mode Commands

The following mode commands, presented in alphabetical order, modify the way the multiple copy commands (CALL, CAD, CAP, CG, CNG) work.

CV	Changes the source and target volumes.
NQ	All members are copied. If you do not set SQ, the multiple copy command defaults to NQ.
RDBK	Does not use the hardware feature but actually reads the data back into storage to verify that it is valid. If you do not set RDBK, the multiple copy command you are using defaults to WV.
ROLLOFF	Turns off roll-screen mode. Then you must press the enter key each time the screen fills up.
ROLLON	Turns on roll-screen mode. In roll-screen mode, you do not need to press the enter key each time the screen fills up. Output "rolls" off the top of the screen as new terminal output appears at the bottom of the screen.
SQ	If you only want to copy some of the members, $COPYUT1 asks you to verify each member before copying it.
WV	Forces a write verify of the target member by using the hardware feature available for validating the data written. WV is the default.

$COPYUT1 - Copy Data Set with Allocation *(continued)*

Using the Copy Commands

With the copy commands you can copy:

- All or selected members in a volume
- All or selected data type members in a volume
- All or selected program types in a volume
- All or selected members beginning with a generic text prefix
- All or selected members that do not begin with a generic text prefix.

If a copy command stops because the target volume on diskette is full, $COPYUT1 issues the following message:

```
DATA4 TOO LARGE TO COPY, ONLY XX RECORDS LEFT IN LIBRARY
TARGET VOL IS FULL, DO YOU WISH TO CONTINUE ON A NEW VOL (Y/N)?
```

If you wish to continue, enter a "Y"; $COPYUT1 issues the following message:

```
MOUNT NEW VOLUME AND DO AN ATTN $VARYON
THEN ENTER ATTN RESTART TO CONTINUE COPY
```

Mount a new volume, vary it online, and continue copying, as follows:

```
> $VARYON 2
EDX001 ONLINE
> RESTART
```

$COPYUT1 - Copy Data Set with Allocation *(continued)*

In this manner, you can create a disk backup spanning several diskettes. Although the copy may take longer using $COPYUT1 instead of $MOVEVOL, you may use fewer diskettes as only members are copied. In addition, you can mix single- and double-sided diskettes. If you are creating a new volume, use $INITDSK (IV command) to start with an empty target volume.

The copy commands will not copy the supervisor ($EDXNUC). This prevents the inadvertent loss of a tailored supervisor. Furthermore, since the supervisor is allocated during disk initialization, the CM command will not allocate $EDXNUC on the target volume. It will copy $EDXNUC from source to target but only if you have allocated the target already and it is the same size as $EDXNUC on the source.

The system does not allow absolute record copy from disk or diskette. Therefore, you cannot use the special names $$, $$EDXLIB, $$EDXVOL. The $COPY utility provides an absolute copy by record number.

To cancel a multiple copy command, press the attention key and enter CA. The command (CALL, CAD, CAP, CG, CNG) ends after the current member is copied.

Note: When using the CAP, CAD, or CALL commands, you can specify the members you want to copy. If the starting member occurs later in the directory list than the ending member, the copy function wraps around and copies all members except those members that occur between the ranges specified.

Each copy command and its explanation is presented in alphabetical order on the following pages.

$COPYUT1 - Copy Data Set with Allocation *(continued)*

CAD — Copy All Data Members from Source to Target

Use CAD to copy data sets designated as D (data) from the source volume to the target volumes. When you allocated data sets using $DISKUT1, you specified one of two organization types: D for data organization or P for program organization. Use data organization to specify data sets used as work files, user source modules, and application data sets. If you use the CAD command, $COPYUT1 only copies those data sets you designated as D (data organization).

You can copy all the data sets or specify a subset of data sets. If you reply Y to the COPY FROM BEGINNING? prompt, $COPYUT1 copies all the data sets. If you respond N to the COPY FROM BEGINNING? prompt, $COPYUT1 prompts you for the first (starting) member and the last (ending) member you want copied.

Example: Copy only data sets designated as D from one volume to another.

```
COMMAND(?):  CAD

COPY FROM BEGINNING?  Y

TEMP       COPY COMPLETE     40 RECORDS COPIED
DATAFILE   COPY COMPLETE    110 RECORDS COPIED
MYDATA     COPY COMPLETE     80 RECORDS COPIED
COMMAND (?):
```

CALL — Copy All Data Sets from Source to Target

Use CALL to copy data sets from the source volume to the target volume. You can copy all the data sets or specify a subset of data sets. If you reply Y to the COPY FROM BEGINNING? prompt, $COPYUT1 copies all the data sets. If you respond N to the COPY FROM BEGINNING? prompt, $COPYUT1 prompts you for the first (starting) member and the last (ending) member you want copied.

When doing a CALL (copy all) function, $COPYUT1 prints the names of the data sets it is copying. When the screen fills up, press the enter key to continue. By specifying the ROLLON command, you turn on roll-screen mode. In roll-screen mode, you do not need to press the enter key after the screen is full. Output "rolls" off the top of the screen as new terminal output appears at the bottom of the screen. You can turn off the roll-screen function by specifying the ROLLOFF command.

$COPYUT1 - Copy Data Set with Allocation *(continued)*

Example 1: Copy all data sets from one volume to another, starting with DATA1 and ending with LASTONE.

```
> $L $COPYUT1
LOADING $COPYUT1     55P,11:16:57, LP= 6900, PART=1

$COPYUT1 - DATA SET COPY UTILITY

              *** WARNING ***
MEMBERS ON TARGET VOLUME WILL BE DELETED.
REALLOCATION AND COPYING OF MEMBERS IS
DEPENDENT ON SUFFICIENT CONTIGUOUS SPACE.

THE DEFINED SOURCE VOLUME IS MYVOL, OK?  Y
THE DEFINED TARGET VOLUME IS MYVOL, OK?  N
ENTER NEW TARGET VOLUME:  YOURVOL
MEMBER WILL BE COPIED FROM MYVOL TO YOURVOL OK?  Y
COMMAND (?):  ROLLON
AUTO ROLL SCREEN MODE ON
COMMAND (?):  CALL

COPY FROM BEGINNING?  N

ENTER STARTING MEMBER:  DATA1
COPY ALL THE MEMBERS AFTER DATA1?  N

ENTER ENDING MEMBER:  LASTONE
DATA1 COPY COMPLETE          256 RECORDS COPIED
DATA2 COPY COMPLETE           12 RECORDS COPIED
  .
  .
LASTONE COPY COMPLETE           14 RECORDS COPIED
COMMAND (?):
```

Example 2: Copy all data sets; SQ command prompts for copy of each data set.

```
COMMAND(?):  SQ
COMMAND(?):  CALL

COPY FROM BEGINNING?  Y

COPY TEMP?  Y
TEMP      COPY COMPLETE     40 RECORDS COPIED
COPY EDITWORK?  N
COPY DATAFILE?  Y
DATAFILE  COPY COMPLETE    110 RECORDS COPIED
COMMAND (?):
```

$COPYUT1 - Copy Data Set with Allocation *(continued)*

CAP — Copy All Programs from Source to Target

Use CAP to copy programs from the source volume to the target volume. When you allocated data sets using $DISKUT1, you specified one of two organization types: D for data organization or P for program organization. Use program organization to specify data sets that contain executable (loadable) Event Driven Executive Language programs. If you use the CAP command, $COPYUT1 only copies those data sets you designated as P (program organization).

You can copy all the data sets or specify a subset of data sets. If you reply Y to the COPY FROM BEGINNING? prompt, $COPYUT1 copies all the data sets. If you respond N to the COPY FROM BEGINNING? prompt, $COPYUT1 prompts you for the first (starting) member and the last (ending) member to be copied.

Example: Copy only programs from one volume to another.

```
COMMAND(?):  CAP

COPY FROM BEGINNING?  Y

PROG1     COPY COMPLETE     40 RECORDS COPIED
PROG2     COPY COMPLETE    110 RECORDS COPIED
PROG3     COPY COMPLETE     80 RECORDS COPIED
LASTPROG  COPY COMPLETE     60 RECORDS COPIED
COMMAND (?):
```

$COPYUT1 - Copy Data Set with Allocation *(continued)*

CG — Copy All Members Starting with a Prefix

Use the CG (copy generic) command to copy only those members beginning with generic text (prefix). $COPYUT1 prompts you for the prefix. $COPYUT1 then searches the source volume directory for names beginning with this prefix and copies only these members to the target volume.

Example: Copy members with prefix of DATA.

```
COMMAND (?):  CG

ENTER GENERIC TEXT:  DATA
DATA1   COPY COMPLETE      54 RECORDS COPIED
DATA2   COPY COMPLETE      13 RECORDS COPIED
DATA3   COPY COMPLETE      50 RECORDS COPIED

DATA4 TOO LARGE TO COPY, ONLY 92 RECORDS LEFT IN LIBRARY
TARGET VOLUME IS FULL, DO YOU WISH TO CONTINUE ON A NEW VOLUME (Y/N)? Y
MOUNT NEW VOLUME AND DO A $VARYON
THEN ENTER ATTN RESTART TO CONTINUE COPY
> $VARYON 2
EDX001 ONLINE
> RESTART

THE DEFINED TARGET VOLUME IS ASMLIB, OK?  Y
VOLUME NOT MOUNTED
TRY AGAIN (Y/N)? Y
ENTER NEW TARGET VOLUME:  EDX001
DATA4     COPY COMPLETE     100 RECORDS COPIED
COMMAND (?):
```

CM — Copy Member from Source to Target

Use CM to copy a member from the source volume to the target volume with the same characteristics. $COPYUT1 automatically allocates the receiving member on the target volume.

$COPYUT1 - Copy Data Set with Allocation *(continued)*

Example 1: Copy a data set (MYPROG) from EDX002 to ASMLIB and rename the data set S1.

```
> $L $COPYUT1
LOADING $COPYUT1     55P,11:16:57, LP= 6900, PART=1

$COPYUT1 - DATA SET COPY UTILITY

               ** WARNING **
MEMBERS ON TARGET VOLUME WILL BE DELETED.
REALLOCATION AND COPYING OF MEMBERS IS
DEPENDENT ON SUFFICIENT CONTIGUOUS SPACE.

THE DEFINED SOURCE VOLUME IS EDX002, OK ?  Y
THE DEFINED TARGET VOLUME IS EDX002, OK ?  N
ENTER NEW TARGET VOLUME:  ASMLIB
MEMBER WILL BE COPIED FROM EDX002 TO ASMLIB OK?:  Y
COMMAND (?):  CM
ENTER FROM(SOURCE) MEMBER:  MYPROG
ENTER TO (TARGET) MEMBER OR * FOR SAME NAME AS SOURCE:  S1
S1    COPY COMPLETE      4 RECORDS COPIED

COMMAND (?):
```

CNG — Copy All Data Sets Not Starting with a Prefix

Use CNG to copy only those data sets that do not begin with the prefix. $COPYUT1 prompts you for a generic text prefix. $COPYUT1 then searches the source volume directory for names that don't begin with the prefix and copies only those data sets to the target volume.

Example: Copy data sets without a prefix of DATA.

```
COMMAND (?):  CNG

ENTER GENERIC TEXT:  DATA
TEMP1   COPY COMPLETE      56 RECORDS COPIED
TEMP2   COPY COMPLETE      15 RECORDS COPIED
TEMP3   COPY COMPLETE      48 RECORDS COPIED

COMMAND (?):
```

EN — End $COPYUT1

Use EN to end the $COPYUT1 utility.

Example: End $COPYUT1.

```
COMMAND (?):  EN

$COPYUT1 ENDED AT 11:18:30
```

$DASDI - Format Disk or Diskette

$DASDI initializes your disks and formats your diskettes.

Invoking $DASDI

You invoke $DASDI with the $L command or option 3.6 of the session manager.

When you invoke $DASDI, it prompts you for one of the following primary initialization options:

- Primary Option 0 - Create a Stand-Alone Dump/$TRAP Diskette
- Primary Option 1 - Diskette Initialization
- Primary Option 2 - 4962 Disk Initialization
- Primary Option 3 - 4963 Disk Initialization
- Primary Option 4 - 4967 Disk Initialization
- Primary Option 5 - DDSK Disk Initialization
- Primary Option 9 - Exit Initialization.

Notes:

1. You can load $DASDI into any partition. $DASDI then loads the initialization routines, $I4962, $I4963, $I4967, or $IDDSK30 into partition 1, and $IDSKETT into any available partition. $DASDI returns an error if partition 1 does not have the space for the initialization routines.
2. When primary options 2, 3, 4, and 5 are executing, do not run a program that accesses the disk being initialized.
3. You can run diskette initialization concurrently with other programs.

$DASDI - Format Disk or Diskette *(continued)*

Primary Option 0 — Create a Stand-Alone

Dump/$TRAP Diskette $TRAP diskette $TRAP diskette $TRAP, creating $TRAP diskette

Primary option 0 uses a 4964, 4965, or 4966 diskette unit to initialize a two-sided, single-density, 256-byte diskette to be used for stand-alone dumps or the $TRAP utility. $DASDI loads a program that places IPL text and the stand-alone dump utility on the front of the diskette. Once you create the diskette, it is ready for use.

If you wish to dump more than 512K bytes, then you must create two diskettes for a stand-alone dump. Once you create them, these two diskettes are identical, and the order in which you use them is not important.

The diskettes you have created are reusable and you do not have to re-create them after you have used them to take a stand-alone dump.

Once you have obtained a stand-alone dump, you can list the contents of the diskette using $DUMP. To dump the contents of the diskette, use data set $$EDXLIB and volume name IBMEDX.

$DASDI - Format Disk or Diskette *(continued)*

Primary Option 0 Example: Create 2 diskettes for a stand-alone dump >512K.

```
> $L $DASDI
LOADING $DASDI        7P,00:34:11, LP= 9D00, PART=1

DIRECT ACCESS DEVICE INITIALIZATION
  DISK INITIALIZATION OPTIONS:
    0 = CREATE STAND-ALONE DUMP/$TRAP DISKETTE
    1 = DISKETTE INITIALIZATION
    2 = 4962 DISK INITIALIZATION
    3 = 4963 DISK INITIALIZATION
    4 = 4967 DISK INITIALIZATION
    5 = DDSK DISK INITIALIZATION
    9 = EXIT DISK INITIALIZATION
  ENTER DISK INITIALIZATION OPTION:  0

  DO YOU WISH TO FORMAT 2 DISKETTES (Y/N)?  Y

DISKETTE TO BE FORMATTED FOR USE WITH:  STAND-ALONE DUMP

IS DISKETTE A 2-SIDED DISKETTE?:  Y

**********************************************
*         DISKETTE FORMATTING PROGRAM        *
*  IF FORMATTING IS IN PROGRESS, DO NOT      *
*  CANCEL ($C) THIS PROGRAM.  INSTEAD, PRESS *
*  ATTN AND ENTER $DASDI TO TERMINATE.       *
**********************************************

ENTER DISKETTE ADDRESS IN HEX:  02
DEVICE VARIED OFFLINE

DEVICE ADDRESS 02, STAND-ALONE DUMP FORMAT, SINGLE DENSITY, 256 BYTES/SECTOR
WARNING :  FORMATTING WILL DESTROY ALL DATA ON THE DISKETTE.
CONTINUE (Y/N)?  Y
IBMEDX VARIED ONLINE

FORMATTING COMPLETE
STAND-ALONE DUMP DISKETTE BUILT

*** REMOVE THE FIRST DISKETTE AND INSERT THE SECOND DISKETTE ***
PRESS "ENTER" TO CONTINUE OR "PF3" TO EXIT OPTION
```

$DASDI - Format Disk or Diskette *(continued)*

Create 2 diskettes for a stand-alone dump >512K (continued).

```
DISKETTE TO BE FORMATTED FOR USE WITH:  STAND-ALONE DUMP

IS DISKETTE A 2-SIDED DISKETTE (Y/N)?  Y
**********************************************
*         DISKETTE FORMATTING PROGRAM        *
*  IF FORMATTING IS IN PROGRESS, DO NOT      *
*  CANCEL ($C) THIS PROGRAM.  INSTEAD, PRESS *
*  ATTN AND ENTER $DASDI TO TERMINATE.       *
**********************************************

ENTER DISKETTE ADDRESS IN HEX:  02
DEVICE VARIED OFFLINE

DEVICE ADDRESS 02, STAND-ALONE DUMP FORMAT, SINGLE DENSITY, 256 BYTES/SECTOR
WARNING :  FORMATTING WILL DESTROY ALL DATA ON THE DISKETTE.
CONTINUE (Y/N)?  Y
IBMEDX VARIED ONLINE

FORMATTING COMPLETE
STAND-ALONE DUMP DISKETTE BUILT

DIRECT ACCESS DEVICE INITIALIZATION
  DISK INITIALIZATION OPTIONS:
    0 = CREATE STAND-ALONE DUMP/$TRAP DISKETTE
    1 = DISKETTE INITIALIZATION
    2 = 4962 DISK INITIALIZATION
    3 = 4963 DISK INITIALIZATION
    4 = 4967 DISK INITIALIZATION
    5 = DDSK DISK INITIALIZATION
    9 = EXIT DISK INITIALIZATION
  ENTER DISK INITIALIZATION OPTION:
```

$DASDI - Format Disk or Diskette *(continued)*

Primary Option 1 — Diskette Initialization

The $DASDI utility initializes single and double-sided diskettes. Three formats are available:

- Format for use with the Series/1 Event Driven Executive
- Format to the IBM Standard for Information Interchange
- Format entire diskette to sector size: 128-, 256-, 512-, or 1024-byte records.

Notes:

1. Double-density is available on the 4965 and 4966 8 inch diskette units at 256, 512, or 1024 bytes per sector. Only double-density at 256 and 1024 bytes per sector is recognized on the Event Driven Executive.
2. 128 bytes per sector are available only at single density on the 8 inch diskette.
3. 1024 bytes per sector are available only at double-density on the 8 inch diskette.

The following matrix shows the configurations available by format for density and sector size when initializing your diskettes.

	128 bytes/sector		256 bytes/sector		512 bytes/sector		1024 bytes/sector	
Formats	**S**	**D**	**S**	**D**	**S**	**D**	**S**	**D**
Event Driven Executive	Y	N/A	Y	Y	N/A	N/A	N/A	Y
Standard for Information Interchange	Y	N/A	Y	Y	Y	Y	N/A	Y
Sector size	Y	N/A	Y	Y	Y	Y	N/A	Y

Legend:
S = Single-density (Diskette 1 and Diskette 2)
D = Double-density (Diskette 2D on 4965 and 4966)

Figure 10. Density and sector sizes available according to format

$DASDI - Format Disk or Diskette *(continued)*

Note: For basic exchange copying under the Standard for Information Interchange format, use a single-sided diskette, formatted as single density, 128 bytes per sector. For H-level exchange, use a double-sided diskette, formatted as double-density, 256 bytes per sector.

Event Driven Executive Format

If you select the Event Driven Executive format, cylinder 0 is formatted according to the IBM Standard for Information Interchange. The remaining cylinders are formatted at 128 or 256 bytes per sector. On a 4965 and a 4966, a diskette may be formatted as double-density at 256 or 1024 bytes per sector.

After surface analysis is complete, $DASDI writes the volume label, IBMEDX, on the diskette. The next step after preparing a diskette surface usually is to create a volume for use with the Event Driven Executive. You create volumes (establish directories) with the $INITDSK utility. $DASDI gives you the option of going directly into $INITDSK execution without having to end $DASDI and issue the $L command for $INITDSK yourself.

Standard for Information Interchange Format

If you select the IBM Standard for Information Interchange, format cylinder 0 according to that standard, and format the remaining cylinders for 128-, 256-, 512-, or 1024-byte records. $DASDI prompts you for the density (single or double) and the sector size (single: 1-128, 2-156, 3-512; or double: 1-256, 2-512, 3-1024). If you are going to use the diskette for basic exchange copy under $COPY, use a single-sided diskette formatted as single density, 128 bytes per sector. If you are going to use the diskette for H-exchange copy using $HXUT1, use a double-sided diskette formatted as double-density at 256 bytes per sector. After surface analysis, $DASDI writes the volume label, IBMEDX, on the diskette and assigns a data set name, DATA. DATA consists of all the tracks on the diskette. Under this format, you do not need to initialize diskettes.

Sector Size Format

If you select the sector size format, $DASDI formats all cylinders to the density (single or double) and sector size you select (128, 256, 512, or 1024 bytes). After surface analysis, $DASDI does not write a volume label, header, or record in cylinder 0, and you are not given the option of going directly into $INITDSK execution.

Note: A diskette initialized according to sector size cannot be used on an Event Driven system except for reformatting.

$DASDI - Format Disk or Diskette *(continued)*

Operating Characteristics

After you invoke $DASDI and choose primary option 1, $DASDI prompts you for the device address of the diskette drive where the diskette to be formatted is mounted. Enter this address in hexadecimal.

Note: The 4966 has a capacity of 23 diskettes: 2 magazines of 10 diskettes each plus 3 slots for individual diskettes. The three individual slots are the first 3 slots in the device. *$DASDI operates on diskettes in slot 1 only*; you must mount in slot 1 any diskette on which you want to run surface preparation.

After you choose a format, $DASDI prompts you (as constrained by format and device choice) for density (single or double) and sector size (128, 256, 512, or 1024 bytes). $DASDI varies the selected diskette device offline, displays the selected format, and issues the following warning message:

```
          *** WARNING ***
     FORMATTING WILL DESTROY ALL DATA
     ON THE DISKETTE.  CONTINUE?
```

If you respond Y, the following occurs for each of the three formats:

1. *Event Driven Executive*—$DASDI formats the diskette, writes a volume label (IBMEDX) on the diskette, and issues the message:

```
FORMATTING COMPLETE
```

   You then have the option of going directly to $INITDSK as follows:

```
LOAD $INITDSK?
```

   If you wish to create a logical volume or establish a directory, respond Y and the system invokes $INITDSK. After you initialize your diskette under $INITDSK, end the $INITDSK utility.

2. *Standard for Information Interchange*—$DASDI formats the diskette and writes a volume label (IBMEDX) on the diskette, allocates a data set called DATA, and issues the following message:

```
FORMATTING COMPLETE
```

   DATA consists of all the data tracks on the diskette.

$DASDI - Format Disk or Diskette *(continued)*

3. *Sector Size Format*—$DASDI formats the diskette but does not write a volume label or header on the diskette; it issues the following message:

```
FORMATTING COMPLETE
```

$DASDI prompts you as follows:

```
ANOTHER DISKETTE?
```

If you respond Y, you have the following choices (you should insert another diskette as required):

Choice 1

```
SAME DEVICE ADDRESS AND FORMAT?  Y
DEVICE VARIED OFFLINE

DEVICE ADDRESS 22, EDX FORMAT, DOUBLE DENSITY, 256 BYTES/SECTOR

                      ******* WARNING ************
FORMATTING WILL DESTROY ALL DATA ON THE DISKETTE IN SLOT 1. CONTINUE?
```

Or

Choice 2

```
SAME DEVICE ADDRESS AND FORMAT?  N
ENTER DISKETTE ADDRESS IN HEX:
```

If you respond N to "ANOTHER DISKETTE," the system displays the $DASDI primary option menu again.

Notes:

1. Do not use $C to cancel a formatting operation. Enter ATTN $DASDI to force termination.

2. After you create the volume label and data set header, the rest of cylinder 0 consists of deleted records. Any attempt to read them results in an error condition.

$DASDI - Format Disk or Diskette *(continued)*

Example 1: Format a double-density diskette on a 4966 for Event Driven Executive.

```
> $L $DASDI
LOADING $DASDI        7P,00:28:55, LP= 7E00, PART=1

DIRECT ACCESS DEVICE INITIALIZATION
  DISK INITIALIZATION OPTIONS:
    0 = CREATE STAND-ALONE DUMP/$TRAP DISKETTE
    1 = DISKETTE INITIALIZATION
    2 = 4962 DISK INITIALIZATION
    3 = 4963 DISK INITIALIZATION
    4 = 4967 DISK INITIALIZATION
    5 = DDSK DISK INITIALIZATION
    9 = EXIT DISK INITIALIZATION
  ENTER DISK INITIALIZATION OPTION:  1

DISKETTE INITIALIZATION

**********************************************
*         DISKETTE FORMATTING PROGRAM        *
*  IF FORMATTING IS IN PROGRESS, DO NOT      *
*  CANCEL ($C) THIS PROGRAM.  INSTEAD, PRESS *
*  ATTN AND ENTER $DASDI TO TERMINATE.       *
**********************************************

ENTER DISKETTE ADDRESS IN HEX:  22

ENTER 4966 SLOT NUMBR (1, 2, 3): 1

INITIALIZE FOR USAGE WITH THE IBM EVENT DRIVEN EXECUTIVE?  (Y/N)? Y

SELECT DENSITY (1=SINGLE, 2=DOUBLE):  2
SELECT SECTOR SIZE (1=256, 2=1024):  1
DEVICE VARIED OFFLINE

DEVICE ADDRESS 22, EDX FORMAT, DOUBLE DENSITY, 256 BYTES/SECTOR

              ** WARNING **
FORMATTING DESTROYS ALL DATA ON THE DISKETTE IN SLOT 1.  CONTINUE?  Y
IBMEDX VARIED ONLINE

FORMATTING COMPLETE
LOAD $INITDSK?  N

ANOTHER DISKETTE?  N
```

$DASDI - Format Disk or Diskette *(continued)*

```
DIRECT ACCESS DEVICE INITIALIZATION
  DISK INITIALIZATION OPTIONS:
    0 = CREATE STAND-ALONE DUMP/$TRAP DISKETTE
    1 = DISKETTE INITIALIZATION
    2 = 4962 DISK INITIALIZATION
    3 = 4963 DISK INITIALIZATION
    4 = 4967 DISK INITIALIZATION
    5 = DDSK DISK INITIALIZATION
    9 = EXIT DISK INITIALIZATION
  ENTER DISK INITIALIZATION OPTION:  9

$DASDI ENDED AT 00:29:30
```

Example 2: Format diskette on a 4964 to IBM Standard for Information Interchange.

```
ENTER DISKETTE ADDRESS IN HEX:  02

INITIALIZE FOR USAGE WITH THE EVENT DRIVEN EXECUTIVE?  N

INITIALIZE TO IBM STANDARDS FOR INFORMATION INTERCHANGE?  Y

NOTE: EDX SUPPORT FOR IBM STANDARDS FOR INFORMATION INTERCHANGE
 DISKETTES IS LIMITED TO THE DOCUMENTED EDX EXCHANGE UTILITIES
SELECT SECTOR SIZE (1=128, 2=256, 3=512):  1
DEVICE VARIED OFFLINE

DEVICE ADDRESS 02, IBM STANDARD FORMAT, SINGLE DENSITY, 128 BYTES/SECTOR

              ** WARNING **
FORMATTING WILL DESTROY ALL DATA ON THE DISKETTE.  CONTINUE?  Y
IBMEDX VARIED ONLINE

FORMATTING COMPLETE

ANOTHER DISKETTE?  N

$DASDI ENDED AT 00:44:30
```

$DASDI - Format Disk or Diskette *(continued)*

Example 3: Format diskette on a 4966 to 256-byte records (double-density).

```
ENTER DISKETTE ADDRESS IN HEX:  22

INITIALIZE FOR USAGE WITH THE EVENT DRIVEN EXECUTIVE?  N

INITIALIZE TO STANDARDS FOR INFORMATION INTERCHANGE?  N
SELECT DENSITY (1=SINGLE, 2=DOUBLE):  2
SELECT SECTOR SIZE (1=256, 2=512, 3=1024):  1
DEVICE VARIED OFFLINE

DEVICE ADDRESS 22, NONSTANDARD, DOUBLE DENSITY, 256 BYTES/SECTOR

              ** WARNING **
FORMATTING WILL DESTROY ALL DATA ON THE DISKETTE IN SLOT 1.  CONTINUE?
```

$DASDI - Format Disk or Diskette *(continued)*

Primary Option 2 — 4962 Disk Initialization

The disk initialization utility for the 4962 initializes your disk, writes sector addresses on the entire volume, analyzes and locates defective sectors, and assigns alternate sectors. After you initialize the disk, it is ready for use with the Event Driven Executive. For a new disk device, you should perform initialization before you install the Event Driven Executive on it.

When using this primary option, you must select one of two initialization types:

- PI (primary) — initialize a disk for the first time or completely reinitialize the disk.

  Note: This type rewrites the complete disk surface and destroys all data that may have been on the disk.

- AS (alternate sector assignment) — assign alternate sectors without destroying the data currently on the disk.

Using PI Initialization

Use PI to verify and correct sector IDs and to analyze the disk surface to find defective sectors. If the programmer's console is active, the data buffer displays the number of the cylinder $DASDI is initializing currently. If the system finds a defective sector, either on a movable or a fixed head, it assigns an alternate sector from cylinder 1 and $DASDI issues a message. When the system assigns an alternate sector, the sector ID of the defective sector refers to the location of its alternate on cylinder 1. The system marks defective sectors. If a defective sector exists on cylinder 0, the system assigns to the defective sector an alternate sector under the same head on cylinder 0.

Using AS Initialization

Use AS to force the assignment of alternate sectors without destruction of data on the disk. $DASDI tries to move data from the defective sector to its assigned alternate. If data recovery fails, $DASDI issues a message and flags the alternate data field with all one-bits (hexadecimal FFFF). If the system finds an assigned alternate is defective, it marks the alternate as defective and assigns a new alternate. The system attempts data recovery in this case, also.

Note: Use AS only when necessary. Cylinder 1 has a limited number of available alternate sectors. Once the system assigns an alternate sector, you can recover the sector only by writing all sector IDs during a primary initialization.

$DASDI - Format Disk or Diskette *(continued)*

The storage capacity and number of alternate sectors available on cylinder 1 depends on the 4962 model:

	Storage capacity (in bytes)		Alternate
Model	Moveable heads	Fixed heads	sectors
1	9,308,160	-	120
1F	9,308,160	122,880	120
2	9,308,160	-	120
2F	9,308,160	122,880	120
3	13,962,240	-	180
4	13,962,240	-	180

Example 1: Primary initialization of a 4962 disk.

```
> $L $DASDI
LOADING $DASDI    7P,00:28:55, LP=7E00, PART=1

DIRECT ACCESS DEVICE INITIALIZATION
  DISK INITIALIZATION OPTIONS:
    0 = CREATE STAND-ALONE DUMP/$TRAP DISKETTE
    1 = DISKETTE INITIALIZATION
    2 = 4962 DISK INITIALIZATION
    3 = 4963 DISK INITIALIZATION
    4 = 4967 DISK INITIALIZATION
    5 = DDSK DISK INITIALIZATION
    9 = EXIT DISK INITIALIZATION
  ENTER DISK INITIALIZATION OPTION:  2

4962 DISK INITIALIZATION

*******************************************************
*                 WARNING                             *
*    NO USER PROGRAM SHOULD BE RUNNING                *
*    WHILE PERFORMING DISK INITIALIZATION             *
*******************************************************
DISK INITIALIZATION STARTED
ENTER DISK DEVICE ADDRESS-NN (HEX):
03

ENTER INITIALIZATION TYPE - PI OR AS:
PI

ENTER INITIALIZATION MODE
REMOVE PREVIOUS ... DEFECTIVE SECTOR FLAGS? Y/N:  N
```

$DASDI - Format Disk or Diskette *(continued)*

In the previous example, $DASDI prompts for the following:

- Disk or diskette primary initialization option: 1 through 6
- Initialization type: PI for primary or AS for alternate sector
- Initialization mode:
  - N - Retain defective flag byte of each sector ID.
  - Y - Rewrite sector flag IDs and reinitialize the flag byte where possible. Allows you to initialize a disk with invalid sector flags.

Note: Respond Y only if you wish to rewrite all sector IDs. This causes the loss of any IBM factory-assigned defective sector flags. If you respond Y, the following verify operation occurs:

```
FACTORY-MARKED DEFECTIVES MAY BE LOST
IS CHANGE OF REPLY DESIRED? Y/N:  N
```

N Operation will continue with flags considered invalid.

Y A reprompt of the previous message results, allowing you to change the status of the defective flags.

The system repeats the following message for each alternate sector assignment, if any occurs:

```
ALTERNATE SECTOR ASSIGNED FOR ccchss
```

Note: ccchss=the address of the alternate sector assigned.

$DASDI - Format Disk or Diskette *(continued)*

Example 2: Alternate sector assignment on a 4962 disk.

```
> $L $DASDI
LOADING $DASDI     7P,00:28:55, LP=7E00, PART=1

DIRECT ACCESS DEVICE INITIALIZATION
  DISK INITIALIZATION OPTIONS:
    0 = CREATE STAND-ALONE DUMP/$TRAP DISKETTE
    1 = DISKETTE INITIALIZATION
    2 = 4962 DISK INITIALIZATION
    3 = 4963 DISK INITIALIZATION
    4 = 4967 DISK INITIALIZATION
    5 = DDSK DISK INITIALIZATION
    9 = EXIT DISK INITIALIZATION
  ENTER DISK INITIALIZATION OPTION:  2

4962 DISK INITIALIZATION

*******************************************************
*                    WARNING                          *
*       NO USER PROGRAM SHOULD BE RUNNING             *
*       WHILE PERFORMING DISK INITIALIZATION          *
*******************************************************
DISK INITIALIZATION STARTED
ENTER DISK DEVICE ADDRESS - NN (HEX):
03

ENTER INITIALIZATION TYPE - PI OR AS:
AS

ALTERNATE SECTOR MODE
ENTER SECTOR ADDRESS - CCCHSS
```

Cylinder/Head/Sector (ccchss): The address of the sector presumed to be defective. $DASDI assigns an alternate sector on cylinder 1 then tries to move the data from the defective sector to the alternate sector. Alternates on cylinder 0 are located on the same track and head as the defects on cylinder 0. This process may reveal that the sector IDs on cylinder 0 are in an inconsistent condition. Processing continues if possible. You cannot assign an alternate to a defective sector on cylinder 1.

Note: The system always refers to the fixed-head area as cylinder 303. You should consider that this cylinder contains eight heads (zero through seven). To refer to sector five under fixed-head four, specify 303405.

The system displays the following message at your terminal indicating completion of the disk initialization.

```
ALTERNATE SECTOR ASSIGNED FOR CCCHSS
DISK INITIALIZATION COMPLETE

$DASDI   ENDED AT 00:31:15
```

$DASDI - Format Disk or Diskette *(continued)*

Primary Option 3 — 4963 Disk Initialization

The $DASDI utility identifies and restores defective sectors on a 4963 disk device. The 4963 comes from the factory already formatted with all logical sector addresses assigned and tested and with alternates assigned to any defective sectors; you do not have to initialize a newly installed 4963.

With this primary option, you can:

- Identify a specific sector as being defective and cause the utility to assign an alternate to it.
- Restore a previously identified defective sector and cause the utility to free its alternate.
- Print a map of all defective sectors and indicate if the defective sector were factory- or user-identified.

The system assigns alternate sectors as follows:

- If the primary alternate (the extra sector on the same track) is available, the system uses it as the alternate for the defective sector.
- If the primary alternate is not available (either it is defective or already assigned), the system assigns a secondary alternate from the nearest track under the movable heads having an available primary alternate.

Note: The system assigns the primary alternate under a fixed head to a sector that is under the same fixed head.

The storage capacity and the number of alternate sectors for each 4963 model follows:

Model	Storage capacity (in bytes)	Alternate sectors
Moveable head:		
23A,B	23,592,960	358
29A,B	29,491,200	358
58A,B	58,982,400	358
64A,B	64,880,640	358
Fixed-head:		
23A, 23B, 58A, 58B only	131,072	358

$DASDI - Format Disk or Diskette *(continued)*

When restoring sectors from defective status, $DASDI physically moves the sectors within the track to minimize the processing time between consecutive logical sectors. You cannot restore:

- A factory-assigned defective sector
- A primary defect (one that causes the system to assign the primary alternate for the track)
- A sector whose ID has been extended (caused by a defect in the ID field of the original sector). (See example 2 on the following page.)

Example 1: Invoking 4963 disk initialization.

```
> $L $DASDI
LOADING $DASDI     7P,00:28:55, LP=7E00, PART=1

DIRECT ACCESS DEVICE INITIALIZATION
  DISK INITIALIZATION OPTIONS:
    0 = CREATE STAND-ALONE DUMP/$TRAP DISKETTE
    1 = DISKETTE INITIALIZATION
    2 = 4962 DISK INITIALIZATION
    3 = 4963 DISK INITIALIZATION
    4 = 4967 DISK INITIALIZATION
    5 = DDSK DISK INITIALIZATION
    9 = EXIT DISK INITIALIZATION
ENTER DISK INITIALIZATION OPTION:  3

4963 DISK INITIALIZATION

*******************************************************
*                WARNING                              *
*     NO USER PROGRAM SHOULD BE RUNNING               *
*      WHILE PERFORMING DISK INITIALIZATION           *
*******************************************************

DISK INITIALIZATION STARTED
ENTER DISK DEVICE ADDRESS-NN (HEX):
48

THE AVAILABLE OPTIONS ARE:
1 - IDENTIFY DEFECTIVE SECTOR(S)
2 - RESTORE DEFECTIVE SECTOR(S)
3 - MAP DEFECTIVE SECTOR(S)
4 - EXIT UTILITY

ENTER OPTION:
```

Your option entry must be one of the four secondary options listed in the command menu. You can choose to identify, restore, or map defective sectors. $DASDI ends when you enter primary option 9.

$DASDI - Format Disk or Diskette *(continued)*

Example 2: Obtaining a map of defective 4963 sectors.

```
ENTER OPTION:
3

DEFECT     ALTERNATE     EXTENDED
0000101
0020114
0340401    3580400
```

This map shows three defective sectors, one of which is a secondary defect (indicated by the alternate address). If the system finds a defective ID and is able to extend that sector to an alternate one, the map displays an asterisk (*) in the EXTENDED field for that sector.

Example 3: Assigning an alternate sector.

```
ENTER OPTION:
1

ENTER CCCHHSS OF SECTOR TO BE MARKED DEFECTIVE OR END:
0010205

ENTER 'Y' TO ASSIGN ALTERNATE, ANYTHING ELSE TO CANCEL:
Y

3580101 ASSIGNED ALTERNATE OF 0010205

ENTER CCCHHSS OF SECTOR TO BE MARKED DEFECTIVE OR END:
END

ENTER OPTION:
```

$DASDI - Format Disk or Diskette *(continued)*

Notes:

1. In the preceding example, enter the disk address for a 4963 as a seven-digit number (0010205): the cylinder is 1 (001), the head is 2 (02), and the sector is 5 (05).

2. $DASDI uses the following range of values for the ccchhss of a 4963:

```
  cylinder     0 - 357
  head         0 - 4 (5,10,11)*
  sector       0 - 63
* depending on model
```

3. $DASDI may appear to assign an alternate for a ccchhss that is not the one you specified. This occurs because the 4963 is arranged as follows:

```
records 0,32 are located in physical sector 0
records 1,33 are located in physical sector 1
  .
  .
  .
records 31,63 are located in physical sector 31
```

Example 4: Restoring a previously assigned alternate sector.

```
ENTER OPTION:
2

ENTER CCCHHSS OF SECTOR TO BE RESTORED OR END:
0010207

0010207 HAS BEEN RESTORED FROM ccchhss

ENTER CCCHHSS OF SECTOR TO BE RESTORED OR END:
END

ENTER OPTION:  4
DISK INITIALIZATION ENDED
$I4963   ENDED AT 01:01:27
```

Note: The system always refers to the fixed-head area on the 4963 as cylinder 511. This cylinder contains 8 heads (16-23) and 64 sectors (0-63).

$DASDI - Format Disk or Diskette *(continued)*

Primary Option 4 — 4967 Disk Initialization

The $DASDI utility identifies and restores defective sectors on a 4967 disk device. However, unlike the 4962 and 4963 disks, the system identifies the sector addresses on the 4967 by relative block address (RBA) rather than by cylinder, track, and sector.

With primary option 4, you can:

- Verify all data fields and associated IDs on the entire disk or a selected cylinder and identify any defective RBAS.
- Refresh data associated with a specified RBA.
- Assign an alternate sector for a specified RBA.
- List the assigned alternate sectors.
- Remove an alternate sector assignment.
- Write one sector ID.

The storage capacity and the number of alternate sectors for each 4967 model follows:

Model	Storage capacity (in bytes)	Alternate sectors
02CA	200,177,152	7,966
02CB	200,177,152	7,966

$DASDI - Format Disk or Diskette *(continued)*

Example 1: Invoking 4967 disk initialization.

```
> $L $DASDI
 LOADING $DASDI     7P,00:28:55, LP=7E00, PART=1

 DIRECT ACCESS DEVICE INITIALIZATION
   DISK INITIALIZATION OPTIONS:
     0 = CREATE STAND-ALONE DUMP/$TRAP DISKETTE
     1 = DISKETTE INITIALIZATION
     2 = 4962 DISK INITIALIZATION
     3 = 4963 DISK INITIALIZATION
     4 = 4967 DISK INITIALIZATION
     5 = DDSK DISK INITIALIZATION
     9 = EXIT DISK INITIALIZATION
 ENTER DISK INITIALIZATION OPTION: 4

 4967 DISK INITIALIZATION

 ********************************************************
 *                  WARNING                             *
 *    NO USER PROGRAM SHOULD BE RUNNING                 *
 *     WHILE PERFORMING DISK INITIALIZATION             *
 ********************************************************

 DISK INITIALIZATION STARTED
 ENTER DEVICE ADDRESS-NN (HEX):
072

 THE AVAILABLE OPTIONS ARE:
 0 - VERIFY ENTIRE DISK
 1 - VERIFY CYLINDER BY SELECTED RBA
 2 - REFRESH DATA
 3 - ASSIGN ALTERNATE SECTOR
 4 - REMOVE ALTERNATE SECTOR ASSIGNMENT
 5 - LIST ALL USER ASSIGNED ALTERNATES
 6 - WRITE ONE SECTOR ID
 7 - EXIT

 ENTER OPTION:
```

Your option entry must be one of the seven secondary options listed in the command menu. Primary option 4 ends when you enter secondary option 7. Once you return to the primary menu, $DASDI ends when you enter primary option 9.

$DASDI - Format Disk or Diskette *(continued)*

Secondary Option 0 — Verify Entire Disk

Use secondary option 0 to identify any defective RBAs on the entire 4967 disk. The system reads and verifies all data fields and associated sector IDs on the disk. If it finds no errors, $DASDI issues the following message:

```
ENTER OPTION:  0
FILE BEING VERIFIED, ID'S AND DATA
NO SECTOR OR DATA ERRORS
```

When you use secondary option 0, no errors are expected to occur. However, if the system detects errors, you should correct them.

If the system detects errors, $DASDI displays a table showing the relative block address, the sector ID of the RBA, the head and cylinder, and a comment describing the error. Following the table, $DASDI displays the recommended ways to correct the errors.

```
 RBA      ID-FG/SEC    HD/CYL      PHY/BSD

00000     1000         0400        03FF NRF-ID NOT VALID
002AE     1000         0400        03FF NRF-ID NOT VALID
BEED9     0001         063A        03FF DATA ERROR
BEEDA     0002         0A3A        03FF SOFT ERR-ECC CORRECTED
BEED8     0000         023A        03FF DATA ERR-ECC INVERTED
BEEDB     BBBB         BBBB        BBBB NRF- ID NOT READABLE
BEEDC     0004         9999        03FF NRF- ID NOT VALID
BEEDD     0006         163A        03FF SOFT ERROR RETRY
BF189     BBBB         BBBB        BBBB NRF- ID NOT READABLE
BF18A     BBBB         BBBB        BBBB NRF- ID NOT READABLE


   UNREADABLE ID- ASSIGN AN ALTERNATE- USE ONLY ONE RBA PER SECTOR
   DATA ERROR- REFRESH RECORD (OPTION 2).  IF ERROR
   PERSISTS, ASSIGN AN ALTERNATE.
   SOFT ERROR- ECC CORRECT- REFRESH RECORD (OPTION 2).
   IF ERROR PERSISTS, ASSIGN AN ALTERNATE.
   ECC INVERTED- REFRESH RECORD (OPTION 2).
   IF ERROR PERSISTS, ASSIGN AN ALTERNATE.
   INVALID ID- WRITE SECTOR ID (OPTION 6).  IF POSSIBLE, DATA
   WILL BE RECOVERED.
```

This table shows flagged or bad sectors that exist on the disk.

Note: Use of secondary option 0 causes all system processing to stop until secondary option 0 completes. This option takes approximately 6 minutes to complete since it is reading and writing every sector on the entire disk.

If you suspect you have a defective RBA (as indicated by disk error messages), you may want to look at your $SYSLOG (cycle steal status) to find which RBA is causing the disk errors. If you do this, then use secondary option 1 to verify that particular RBA.

$DASDI - Format Disk or Diskette *(continued)*

Secondary Option 1 — Verify a Cylinder by Selected RBA

Use secondary option 1 to identify defective RBAs on a particular cylinder. The system reads and verifies all data fields and associated IDs on the selected cylinder. If it finds no errors, $DASDI issues the following message:

```
ENTER OPTION: 1
ENTER RELATIVE BLOCK ADDRESS (RBA)-(HEX) 5 DIGITS (RRRRR): 12345
NO SECTOR OR DATA ERRORS
```

If $DASDI finds an error, it displays a table showing the relative block address, the sector ID of the RBA, the head and cylinder, and a comment describing the error. Following the table, $DASDI displays the recommended ways to correct the errors.

```
 RBA      ID-FG/SEC     HD/CYL    PHY/BSD

00000     1000          0400      03FF NRF-ID NOT VALID
002AE     1000          0400      03FF NRF-ID NOT VALID
BEED9     0001          063A      03FF DATA ERROR
BEEDA     0002          0A3A      03FF SOFT ERR-ECC CORRECTED
BEED8     0000          023A      03FF DATA ERR-ECC INVERTED
BEEDB     BBBB          BBBB      BBBB NRF- ID NOT READABLE
BEEDC     0004          9999      03FF NRF- ID NOT VALID
BEEDD     0006          163A      03FF SOFT ERROR RETRY
BF189     BBBB          BBBB      BBBB NRF- ID NOT READABLE
BF18A     BBBB          BBBB      BBBB NRF- ID NOT READABLE


   UNREADABLE ID- ASSIGN AN ALTERNATE- USE ONLY ONE RBA PER SECTOR
   DATA ERROR- REFRESH RECORD (OPTION 2).  IF ERROR
   PERSISTS, ASSIGN AN ALTERNATE.
   SOFT ERROR- ECC CORRECT- REFRESH RECORD (OPTION 2).
   IF ERROR PERSISTS, ASSIGN AN ALTERNATE.
   ECC INVERTED- REFRESH RECORD (OPTION 2).
   IF ERROR PERSISTS, ASSIGN AN ALTERNATE.
   INVALID ID- WRITE SECTOR ID (OPTION 6).  IF POSSIBLE, DATA
   WILL BE RECOVERED.
```

This table shows flagged or bad sectors that exist on a selected cylinder.

$DASDI - Format Disk or Diskette *(continued)*

Secondary Option 2 — Refresh Data

Use secondary option 2 to refresh the data contained in the sector to ensure that it is valid or correct. You are able to patch data in the suspected RBA and write the new data back out to the RBA. If the error no longer appears, you can use the RBA as is. If the error still appears, then you must assign an alternate sector (secondary option 3).

Notes:

1. For inverted ECC errors, if you write back the RBA without correcting it, the inverted ECC error disappear. However, the data could still be bad. Be sure to verify the data.

2. For ECC corrected errors, you need not verify the data before writing it back to disk.

Example:

```
ENTER OPTION:  2
ENTER RELATIVE BLOCK ADDRESS (RBA)-(HEX) 5 DIGITS (RRRRR):  12345
```

At this point, $DASDI reads the selected RBA and displays the 256 bytes, in eight-words-per-line format, contained in the RBA.

```
D368: 0000 23AB 0000 23AB 0000 23AB 0000 23AB
D378: 0000 23AB 0000 23AB 0000 23AB 0000 23AB
D388: 0000 23AB 0000 23AB 0000 23AB 0000 23AB
D398: 0000 23AB 0000 23AB 0000 23AB 0000 23AB
D3A8: 0000 23AB 0000 23AB 0000 23AB 0000 23AB
D3B8: 0000 23AB 0000 23AB 0000 23AB 0000 23AB
D3C8: 0000 23AB 0000 23AB 0000 23AB 0000 23AB
D3D8: 0000 23AB 0000 23AB 0000 23AB 0000 23AB
D3E8: 0000 23AB 0000 23AB 0000 23AB 0000 23AB
D3F8: 0000 23AB 0000 23AB 0000 23AB 0000 23AB
F408: 0000 23AB 0000 23AB 0000 23AB 0000 23AB
F418: 0000 23AB 0000 23AB 0000 23AB 0000 23AB
```

$DASDI then prompts you as follows:

```
PATCH THE RBA IN STORAGE?
```

If you enter Y, $DASDI issues a prompt for the starting address and number of words (count) of the patch:

```
ENTER THE ADDRESS,COUNT:  D368,1
```

$DASDI - Format Disk or Diskette *(continued)*

$DASDI then displays the address of the area you want to patch and the data currently appearing at that location. Enter the new data.

```
D368:0000
DATA:FFFF
```

$DASDI displays the selected RBA showing the selected address and changed data.

```
D368 FFFF 23AB 0000 23AB 0000 23AB 0000 23AB
D378 0000 23AB 0000 23AB 0000 23AB 0000 23AB
D388 0000 23AB 0000 23AB 0000 23AB 0000 23AB
D398 0000 23AB 0000 23AB 0000 23AB 0000 23AB
D3A8 0000 23AB 0000 23AB 0000 23AB 0000 23AB
D3B8 0000 23AB 0000 23AB 0000 23AB 0000 23AB
D3C8 0000 23AB 0000 23AB 0000 23AB 0000 23AB
D3D8 0000 23AB 0000 23AB 0000 23AB 0000 23AB
D3E8 0000 23AB 0000 23AB 0000 23AB 0000 23AB
D3F8 0000 23AB 0000 23AB 0000 23AB 0000 23AB
F408 0000 23AB 0000 23AB 0000 23AB 0000 23AB
F418 0000 23AB 0000 23AB 0000 23AB 0000 23AB
```

$DASDI then prompts if you wish to patch another location in the same RBA.

```
PATCH THE RBA IN STORAGE? N
```

If you respond Y, $DASDI prompts for another address and count. If you respond N, $DASDI issues a prompt asking if you wish to write the changed RBA to disk. If you respond Y to this prompt, $DASDI writes the changed RBA back to the disk.

```
WRITE THE RBA TO DISK? Y
DATA RECORD WRITTEN
```

Once you have written the RBA to disk, $DASDI issues a prompt asking if you wish to patch another area.

```
ANOTHER PATCH? Y/N
```

If you respond Y, $DASDI again prompts for the selected RBA and displays the 256 bytes contained in that RBA in eight-word-per-line format. If you respond N, $DASDI returns to the 4967 Disk Initialization secondary menu.

$DASDI - Format Disk or Diskette *(continued)*

Secondary Option 3 — Assign Alternate Sector

Use secondary option 3 to assign an alternate sector for a selected RBA.

Example:

```
ENTER OPTION:  3
ENTER RELATIVE BLOCK ADDRESS (RBA)-(HEX) 5 DIGITS (RRRRR):  12345
ALTERNATE ASSIGNED
```

Secondary Option 4 — Remove Alternate Sector Assignment

Use secondary option 4 to remove an alternate sector assignment that you assigned using secondary option 3.

Example:

```
ENTER OPTION:  4
ENTER RELATIVE BLOCK ADDRESS (RBA)-(HEX) 5 DIGITS (RRRRR):  12345
ALTERNATE UNASSIGNED
```

If the RBA you specified is not assigned to an alternate sector, $DASDI issues the following prompt:

```
USER DEFECT NOT FOUND
ALTERNATE UNASSIGNMENT NOT COMPLETED
```

Secondary Option 5 — List All User-Assigned Sectors

Use secondary option 5 to display a list of alternate sectors you assigned.

Example:

```
ENTER OPTION:  5

DISK BEING SEARCHED FOR USER DEFECTS

USER ASSIGNED DEFECTIVE SECTORS
   RBA     RBA    ID=FG/SEC HD/CYL PHY/BSD
   12345   125F3     1100    0436    3BFF
```

$DASDI - Format Disk or Diskette *(continued)*

Secondary Option 6 — Write One Sector ID

Use secondary option 6 to convert the selected RBA to a head, cylinder, and logical number. The system writes the sector ID for that logical number using the factory defect data (if any exist) from the surface analysis cylinder. The system recovers and writes data with the sector ID.

Example:

```
ENTER OPTION: 6
ENTER RELATIVE BLOCK ADDRESS (RBA)-(HEX) 5 DIGITS (RRRRR): 12345
SECTOR ID WRITTEN
```

If the system has already assigned the selected RBA as a user-defined alternate, then $DASDI displays the following message:

```
USER-ASSIGNED ALTERNATE SELECTED, ID WILL NOT BE WRITTEN.
```

If you specify the surface analysis cylinder, then the following message is displayed:

```
SURFACE ANALYSIS CYLINDER SPECIFIED
ID/ID'S NOT WRITTEN
```

If the system detects an error while recovering data, it displays the following message:

```
RBA- XXXXX SECTOR WRITTEN.  DATA ERROR.
AN INVERTED ECC HAS BEEN WRITTEN.
```

In response to this message, refresh the data using secondary option 2.

Secondary Option 7 — Exit

Use secondary option 7 to end 4967 initialization (primary option 4) and to return to the primary option menu of $DASDI. Primary option 9 ends the $DASDI utility.

$DASDI - Format Disk or Diskette *(continued)*

Primary Option 5 — DDSK Disk Initialization

The $DASDI utility identifies and restores defective sectors on the 30-megabyte disk device (DDSK-30) and the 60-megabyte disk (DDSK-60). For these devices, the system identifies disk sector addresses by relative block address (RBA) rather than by cylinder, track, and sector.

With this option, you can:

- Verify all data fields and associated IDs on the entire disk or a selected cylinder.
- Identify any defective RBAs.
- Refresh data associated with a specific RBA.
- Assign an alternate sector for a specified RBA.
- List the assigned alternate sectors.
- Remove an alternate sector assignment.

The initialization routines for the DDSK-30 and DDSK-60 disks appear as primary option 5 under the $DASDI utility. When you load $DASDI, it displays the following menu:

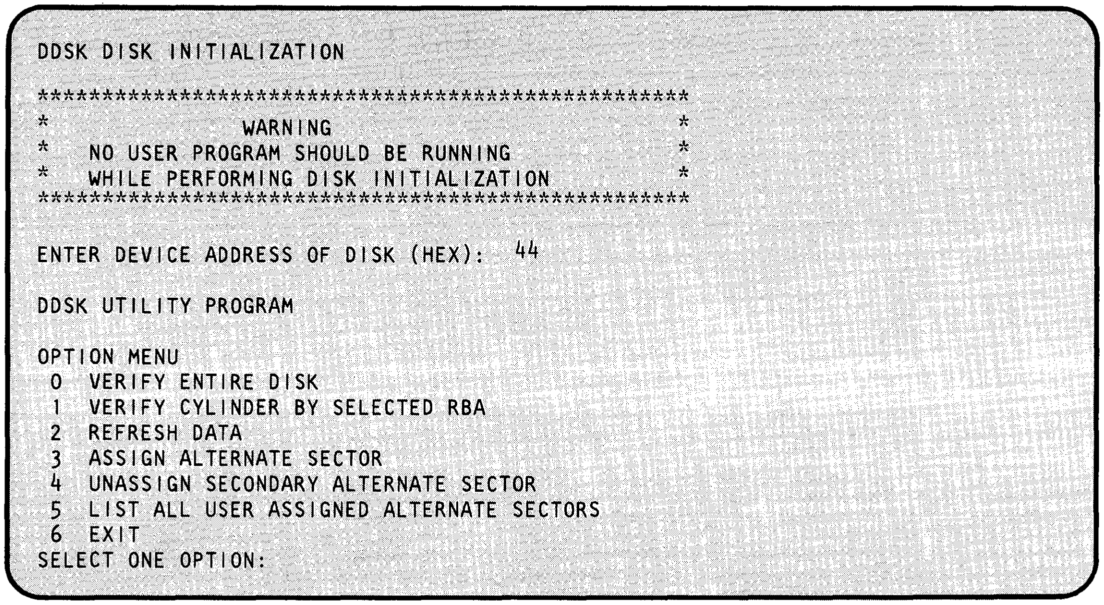
```
> $L $DASDI
LOADING $DASDI       7P,00:34:11, LP=9D00, PART=1

DIRECT ACCESS DEVICE INITIALIZATION
  DISK INITIALIZATION OPTIONS:
    0 = CREATE STAND-ALONE DUMP/$TRAP DISKETTE
    1 = DISKETTE INITIALIZATION
    2 = 4962 DISK INITIALIZATION
    3 = 4963 DISK INITIALIZATION
    4 = 4967 DISK INITIALIZATION
    5 = DDSK DISK INITIALIZATION
    9 = EXIT DISK INITIALIZATION
  ENTER DISK INITIALIZATION OPTION: 5
```

The storage capacity and the number of alternate sectors for DDSK-30 and DDSK-60 disks follows:

Model	Storage capacity (in bytes)	Alternate sectors
30D	30,821,888	3,544
60D	61,668,864	7,088

$DASDI - Format Disk or Diskette *(continued)*

Enter a 5 in response to the ENTER DISK INITIALIZATION OPTION prompt. $DASDI then prompts you for the address of the disk. Once you have entered the disk address, $DASDI displays the secondary options available under option 5.



```
DDSK DISK INITIALIZATION

***************************************************
*               WARNING                           *
*   NO USER PROGRAM SHOULD BE RUNNING             *
*   WHILE PERFORMING DISK INITIALIZATION          *
***************************************************

ENTER DEVICE ADDRESS OF DISK (HEX):  44

DDSK UTILITY PROGRAM

OPTION MENU
 0  VERIFY ENTIRE DISK
 1  VERIFY CYLINDER BY SELECTED RBA
 2  REFRESH DATA
 3  ASSIGN ALTERNATE SECTOR
 4  UNASSIGN SECONDARY ALTERNATE SECTOR
 5  LIST ALL USER ASSIGNED ALTERNATE SECTORS
 6  EXIT
SELECT ONE OPTION:
```

Select one of the six secondary options listed above. A description of each secondary option follows.

$DASDI - Format Disk or Diskette *(continued)*

Secondary Option 0 — Verify Entire Disk

Use secondary option 0 to identify any defective RBAs on the entire DDSK-30 or DDSK-60 disks. The system reads and verifies all data fields and associated sector IDs on the disk. If $DASDI finds no errors, it issues the following message:

```
SELECT ONE OPTION:  0

ENTIRE DISK BEING VERIFIED, IDS AND DATA
    NO SECTOR OR DATA ERRORS
```

When using secondary option 0, no errors are expected to occur. However, if the system detects errors, you should correct them. $DASDI displays the relative block address, the sector ID of the RBA, the head and cylinder, and a comment describing each error. Following the error description, $DASDI displays the recommended way to correct each error.

```
ENTIRE DISK BEING VERIFIED, IDS AND DATA
ERROR SECTORS
     RBA       ID-FG/SEC     HD/CYL

     01222        3244         0011       NRF - ID NOT VALID
     11111        3244         0101       NRF - ID NOT READABLE
     11111        3244         0101       DATA ERROR
     12346        3645         0D02       SOFT ERR - ECC CORRECTED
     BEED8        0000         023A       DATA ERR - ECC INVERTED

RECOMMENDED WAY TO CORRECT LISTED ERRORS:

    NRF - ID NOT VALID:  NO RECORD FOUND - ID NOT READABLE - ASSIGN
    AN ALTERNATE.
    NRF - ID NOT READABLE:  NO RECORD FOUND - ID NOT READABLE - ASSIGN
    AN ALTERNATE.
    DATA ERROR:  READ AND CORRECT RECORD. USE OPTION 2 TO REFRESH
    RECORD. IF ERROR PERSISTS, ASSIGN AN ALTERNATE.
    DATA ERROR:  CORRECTED ECC. USE OPTION 2 TO REFRESH RECORD.
    IF ERROR PERSISTS, ASSIGN AN ALTERNATE.
    DATA ERROR:  INVERTED ECC. USE OPTION 2 TO REFRESH RECORD.
    IF ERROR PERSISTS, ASSIGN AN ALTERNATE.
```

Note: Use of secondary option 0 causes all system processing to stop until secondary option 0 completes. This option takes approximately 6 minutes to complete since it is reading and writing every sector on the disk.

If you suspect that there is a defective RBA (as indicated by a disk error message), use secondary option 1 to verify that particular RBA.

$DASDI - Format Disk or Diskette *(continued)*

Secondary Option 1 — Verify a Cylinder by Selected RBA

Use secondary option 1 to identify defective RBAS on a particular cylinder. The system reads and verifies all data fields and associated IDs on the selected cylinder. If $DASDI finds no errors, it issues the following message:

```
SELECT ONE OPTION:  1

ENTER RELATIVE BLOCK ADDRESS (RBA) 5 HEX DIGITS (RRRRR):  12456

SELECTED CYLINDER BEING VERIFIED, IDS AND DATA
    NO SECTOR OR DATA ERRORS
```

$DASDI displays the relative block address, the sector ID of the RBA, the head and cylinder, and a comment describing any error it finds. Following the error description, $DASDI displays the recommended way to correct the error.

```
ERROR SECTORS
     RBA   ID-FG/SEC   HD/CYL
     11111     3645     0D02      DATA ERR-ECC INVERTED


RECOMMENDED WAY TO CORRECT LISTED ERRORS:
DATA ERROR:  INVERTED ECC
USE OPTION 2 TO REFRESH RECORD.  IF ERROR PERSISTS, ASSIGN AN ALTERNATE.
```

Secondary Option 2 — Refresh Data

Use secondary option 2 to refresh the data contained in the sector to ensure it is valid or correct. You are able to patch data in the suspected RBA and write the new data back out to the RBA. If the error disappears, the RBA can be used as is. If the error remains, then you must assign an alternate sector (secondary option 3).

Notes:

1. For inverted ECC errors, if you write the RBA back without correcting the error, the inverted ECC error disappears. However, the data could still be bad. Be sure to verify the data.

2. For ECC corrected errors, you don't need to verify the data before writing it back to disk.

Example:

```
SELECT ONE OPTION:  2

ENTER RELATIVE BLOCK ADDRESS (RBA)-5 HEX DIGITS (RRRRR):  12346
```

$DASDI - Format Disk or Diskette *(continued)*

At this point, $DASDI reads the selected RBA and displays the 256 bytes (1 record), in eight-words-per-line format, contained in the RBA.

```
0000: 0000 0000 0000 0000 0000 0000 0000 0000
0010: 0000 0000 0000 0000 0000 0000 0000 0000
0020: 0000 0000 0000 0000 0000 0000 0000 0000
0030: 0000 0000 0000 0000 0000 0000 0000 0000
0040: 0000 0000 0000 0000 0000 0000 0000 0000
0050: 0000 0000 0000 0000 0000 0000 0000 0000
0060: 0000 0000 0000 0000 0000 0000 0000 0000
0070: 0000 0000 0000 0000 0000 0000 0000 0000
0080: 0000 0000 0000 0000 0000 0000 0000 0000
0090: 0000 0000 0000 0000 0000 0000 0000 0000
00A0: 0000 0000 0000 0000 0000 0000 0000 0000
00B0: 0000 0000 0000 0000 0000 0000 0000 0000
00C0: 0000 0000 0000 0000 0000 0000 0000 0000
00D0: 0000 0000 0000 0000 0000 0000 0000 0000
00E0: 0000 0000 0000 0000 0000 0000 0000 0000
00F0: 0000 0000 0000 0000 0000 0000 0000 0000
```

$DASDI then prompts you as follows:

```
PATCH THE RBA IN STORAGE?
```

If you enter Y, $DASDI issues a prompt for the starting address and number of words (count) of the patch:

```
ENTER ADDRESS,COUNT:  F0 1
```

$DASDI then displays the address of the area you want to patch and the data currently appearing at that location. Enter the new data.

```
00F0: 0000
DATA:  5
```

$DASDI - Format Disk or Diskette *(continued)*

$DASDI displays the selected RBA showing the selected address and changed data.

```
0000: 0000 0000 0000 0000 0000 0000 0000 0000
0010: 0000 0000 0000 0000 0000 0000 0000 0000
0020: 0000 0000 0000 0000 0000 0000 0000 0000
0030: 0000 0000 0000 0000 0000 0000 0000 0000
0040: 0000 0000 0000 0000 0000 0000 0000 0000
0050: 0000 0000 0000 0000 0000 0000 0000 0000
0060: 0000 0000 0000 0000 0000 0000 0000 0000
0070: 0000 0000 0000 0000 0000 0000 0000 0000
0080: 0000 0000 0000 0000 0000 0000 0000 0000
0090: 0000 0000 0000 0000 0000 0000 0000 0000
00A0: 0000 0000 0000 0000 0000 0000 0000 0000
00B0: 0000 0000 0000 0000 0000 0000 0000 0000
00C0: 0000 0000 0000 0000 0000 0000 0000 0000
00D0: 0000 0000 0000 0000 0000 0000 0000 0000
00E0: 0000 0000 0000 0000 0000 0000 0000 0000
00F0: 0005 0000 0000 0000 0000 0000 0000 0000
```

$DASDI then prompts if you wish to patch another area.

```
ANOTHER PATCH?
```

If you respond Y, $DASDI issues the ENTER ADDRESS,COUNT prompt. If you respond N, $DASDI asks if you wish to write the patched RBA to disk.

```
WRITE THE RBA TO DISK?
```

If you respond Y, $DASDI writes the changed RBA back to the disk and prompts if you want to read another RBA.

```
WRITE THE RBA TO DISK?  Y
WRITE DATA RECORD COMPLETED
READ ANOTHER RBA?
```

If you respond N, $DASDI asks if you wish to read another RBA. If you respond Y, $DASDI again prompts for the selected RBA and displays the 256 bytes contained in that RBA in eight-word-per-line format. If you respond N, $DASDI returns to the DDSK Disk Initialization menu.

$DASDI - Format Disk or Diskette *(continued)*

Secondary Option 3 — Assign Alternate Sector

Use secondary option 3 to assign alternate sectors for a selected RBA. When assigning alternate sectors, you should be familiar with the layout of the disk. Each track contains 68 sectors (RBAs) for your use plus up to two sectors for use as alternates. The alternate sectors on the same track are called primary alternate 1 and primary alternate 2.

When an RBA requires an alternate assignment, $DASDI attempts to assign primary alternate 1. If primary alternate 1 has already been assigned by the same RBA or another RBA, $DASDI attempts to assign primary alternate 2. If primary alternate 2 is also unavailable, $DASDI assigns a secondary alternate sector. The secondary alternate sector is always located on a track different than the track where primary alternates 1 and 2 are located.

If a secondary alternate goes bad, you can assign another secondary alternate. However, using secondary option 4, you can unassign only the last secondary alternate that you assigned.

Example:

```
ENTER OPTION:  3
ENTER RELATIVE BLOCK ADDRESS (RBA)-(HEX) 5 DIGITS (RRRRR):  12345
ALTERNATE ASSIGNMENT COMPLETED
```

Secondary Option 4 — Unassign Secondary Alternate Sector

Use secondary option 4 to unassign the last secondary alternate sector that you assigned with secondary option 3. You can only unassign a secondary alternate.

Example:

```
ENTER OPTION:  4
ENTER RELATIVE BLOCK ADDRESS (RBA)-(HEX) 5 DIGITS (RRRRR):  12345
ALTERNATE UNASSIGNMENT COMPLETED
```

If the RBA you specified is not a secondary alternate sector, $DASDI issues the following prompt:

```
NO USER SECONDARY ALTERNATE FOUND
ALTERNATE UNASSIGNMENT NOT COMPLETED
```

$DASDI - Format Disk or Diskette *(continued)*

Secondary Option 5 — List All User-Assigned Alternate Sectors

Use secondary option 5 to display a list of alternate sectors you assigned. $DASDI issues a listing of all the primary and secondary alternates assigned.

Example:

```
SELECT ONE OPTION:  5

USER ASSIGNED ALTERNATE SECTORS
     RBA     ID-FG/SEC  HD/CYL
     01222        3244    0011     PRIMARY ALTERNATE 1
     01222        3245    0011     PRIMARY ALTERNATE 2
     11111        3244    0101     PRIMARY ALTERNATE 1
     11111        3245    0101     PRIMARY ALTERNATE 2
     11111        3645    0D02     SECONDARY ALTERNATE
     12346        3244    0112     PRIMARY ALTERNATE 1
     12346        3245    0112     PRIMARY ALTERNATE 2
```

The ID-FG/SEC column contains two bytes. The first byte in this column is a flag byte which indicates the condition of the sector's surface. The second byte is a sector byte which is the hexadecimal representation of the alternate sector assigned. For a description of the flag and sector bytes, refer to one of the following hardware description manuals:

- *IBM Series/1 4952 Processor Model 30D Processor Features Description,* GA34-0251
- *IBM Series/1 4954 Processor Model 30D and Model 60D and Processor Features Description,* GA34-0252
- *IBM Series/1 4956 Processor Model 30D and Model 60D and Processor Features Description,* GA34-0253
- *IBM Series/1 4965 Storage and I/O Expansion Unit Description,* GA34-0254

Secondary Option 6 — Exit

Use secondary option 6 to end DDSK initialization (primary option 5) and return to the primary option menu of $DASDI. Primary option 9 ends the $DASDI utility.

$DASDI - Format Disk or Diskette *(continued)*

Primary Option 9 — Exit Initialization

Use primary option 9 to end the $DASDI utility.

```
> $L $DASDI
LOADING $DASDI       7P,00:34:11, LP=9D00, PART=1

DIRECT ACCESS DEVICE INITIALIZATION
  DISK INITIALIZATION OPTIONS:
    0 = CREATE STAND-ALONE DUMP/$TRAP DISKETTE
    1 = DISKETTE INITIALIZATION
    2 = 4962 DISK INITIALIZATION
    3 = 4963 DISK INITIALIZATION
    4 = 4967 DISK INITIALIZATION
    5 = DDSK DISK INITIALIZATION
    9 = EXIT DISK INITIALIZATION
  ENTER DISK INITIALIZATION OPTION:  9

$DASDI ENDED AT 01:12:29
```

$DEBUG - Debugging Tool

Use $DEBUG to locate errors in programs. All of your interactions are through terminals; none require the use of the optional programmer console. By operating a program under control of $DEBUG, you can:

- Stop the program each time execution reaches an instruction address (breakpoint), that you have specified.
- Trace the flow of execution of instructions within the program by specifying one or more ranges of instruction addresses known as trace ranges. Each time the program executes an instruction within any of the specified trace ranges, the terminal displays a message identifying the task name and the instruction address just executed. You may stop program execution after the system executes each instruction within a trace range. You may restart program execution at other than the next instruction.
- List additional registers and storage location contents while the program is stopped at a breakpoint or at an instruction within a trace range.
- Patch the contents of storage locations and registers.
- Restart program execution at the breakpoint or trace range address where it is currently stopped.

You can determine the results of computations performed by the program and the sequence of instruction execution within the program. You can also modify data or instructions of the program during execution.

Major Features of $DEBUG

A summary of the major features of the $DEBUG program follows:

- You can establish multiple breakpoints and trace ranges.
- You can debug overlay segments.
- Up to five users can employ separate copies of $DEBUG concurrently if the system has sufficient storage available.
- You can set breakpoints and trace ranges in the Series/1 assembler language as well as in Event Driven Language instructions.
- The system automatically obtains the task names from the program you are testing.
- You can display and modify task registers #1 and #2.
- You can display and modify hardware registers R0 through R7 and the IAR.
- You can display and modify the task return code words.

$DEBUG - Debugging Tool *(continued)*

- The list and patch functions accept five different data formats.
- You can run the program you are debugging in a partition other than the one where $DEBUG is loaded.
- You can make all address specifications as shown in the program assembly listing without concern for the actual storage addresses where the system loads the program into storage for testing.
- The task where you set the trace ranges is the only place that incurs processing overhead. Even then, the system enables the hardware trace feature only for specific tasks.
- You can activate the debug facility for a program that is experiencing problems even if you previously loaded that program without the debug facility.
- You can debug a program by loading $DEBUG from a terminal other than the one from which you loaded the program you are testing.
- You can debug a program that uses a 4978, 4979, 4980, or 3101 terminal in STATIC or ROLL screen mode.
- You can list breakpoints or trace ranges specified during a debug session.
- $DEBUG can control the execution of programs containing up to 20 tasks.

Necessary Data for Debugging

To use $DEBUG, you must include $DBUGNUC at system generation. You must have a printed listing of the program you are debugging that shows the storage addresses of each instruction and data area of interest. To obtain such a listing using $S1ASM, specify PRINT GEN in the source program, after the PROGRAM statement, at assembly time. Precede the PROGRAM statement with PRINT NOGEN to prevent the system from printing many system EQU statements, among others. For $EDXASM, you can get a satisfactory listing if you specify LIST.

To debug segment overlays, you need a link map to find the overlay segment numbers.

For an example of debugging an application program, see the *Event Driven Executive Language Programming Guide.*

$DEBUG - Debugging Tool *(continued)*

Invoking $DEBUG

Invoke $DEBUG with the $L operator command. The session manager does not support this utility.

```
> $L $DEBUG
LOADING $DEBUG     31P,09:37:17, LP=C200, PART=1
```

$DEBUG then prompts for the name of the program you wish to debug as follows:

```
PROGRAM (NAME,VOLUME):  DBUGDEMO,EDX002
```

The program that you are debugging does not have to run in the same partition where the system loaded $DEBUG. After you enter the program name, the system prompts you for a partition number. The system prompts you for a terminal name only if $DEBUG is loading your program. (You may enter the partition number and terminal name on the same line as the program name if you prefer.)

```
PARTITION (DEFAULT IS CURRENT PARTITION):  3
TERMINAL NAME (DEFAULT IS CURRENT TERMINAL):  $SYSLOG
```

The system will inform you if there is not enough room for your program in the partition you specified or if you specified an invalid partition number. You may specify 0 as a partition number to tell $DEBUG to load your program in the first available partition. $DEBUG will not look to see if your program is already in storage.

Note: Do not enter the name of a printer when prompted for the name of the terminal on which $DEBUG is to load your application program.

After you enter the partition number and the terminal name, if applicable, $DEBUG displays the load point of the program as follows:

```
LOADING DBUGDEMO     4P,09:10:28, LP= 2000,PART= 3
```

If you have loaded the program you are debugging into storage multiple times, the system lists the load points of all currently active copies as follows:

```
ALREADY ACTIVE AT 5200 5600 5A00
```

$DEBUG - Debugging Tool *(continued)*

You have the option of requesting a fresh copy of the program or specifying which copy of the program you wish to debug.

```
DO YOU WANT A NEW COPY TO BE LOADED?
```

If you respond "Y," $DEBUG loads a new copy of the program.

```
LOADING DBUGDEMO   4P, 09:38:02, LP= 5E00
```

If you respond "N," $DEBUG prompts you for the load point of the copy you wish to debug.

```
PROGRAM LOAD POINT:  5600
LOADING DBUGDEMO  4P, 09:38:02, LP= 5600
```

$DEBUG Commands

To display the $DEBUG commands at your terminal, press the attention key and enter the HELP command as follows:

```
> HELP

HELP     - LIST DEBUG COMMANDS
WHERE    - DISPLAY TASK STATUS
LIST     - DISPLAY STORAGE OR REGISTERS
PATCH    - MODIFY STORAGE OR REGISTERS
QUALIFY  - MODIFY BASE ADDR
AT       - ESTABLISH BREAKPOINTS
OFF      - REMOVE BREAKPOINTS
GO       - START TASK PROCESSING
POST     - POST EVENT OR PROCESS INTERRUPT
PRINT    - DIRECT LISTING TO PRINTER
BP       - LIST BREAK POINTS
GOTO     - CHANGE EXECUTION SEQUENCE
CLOSE    - CLOSE SPOOL JOB CREATED BY $DEBUG
END      - TERMINATE DEBUG FACILITY
```

You specify each command by pressing the attention key on your terminal and entering the command name or the command name plus the required parameters for the command.

$DEBUG - Debugging Tool *(continued)*

Syntax Summary

The following examples show the various formats of the AT command. Example 1 shows interactive mode and example 2 shows single-line entry. Syntax examples for each command capitalize the $DEBUG command keyword parameters and show the variable parameters in lowercase. A slash (/) separates the keyword options that you can specify.

Example 1:

```
> AT
OPTION(*/ADDR/TASK/ALL):  TASK
TASK NAME:  TIMET
LOW ADDRESS:  0
HIGH ADDRESS:  3000
LIST/NOLIST:  LIST
OPTION(*/ADDR/R0...R7/#1/#2/IAR/TCODE/UNMAP):  #1
TASK NAME:  LOOP2
LENGTH:  2
MODE(X/F/D/A/C/E/L):  F
STOP/NOSTOP:  NOSTOP
      1 BREAKPOINT(S) SET
```

Example 2: You can obtain identical results by entering the single response. However, when using single-line entry, be sure that you enter the parameters in the order $DEBUG expects them.

```
> AT T TIMET 0 3000 L #1 LOOP2 2 F N
1 BREAKPOINT(S) SET
```

Each command with its syntax description follows in alphabetical order.

AT — Set Breakpoints and Trace Ranges

AT sets breakpoints and trace ranges. The system executes the LIST and STOP options established for a breakpoint or trace range prior to executing the instruction that satisfied the breakpoint or trace range specification. When the system satisfies the specification for a breakpoint or trace range, it reroutes the currently active task. Then $DEBUG performs the following actions for the subject task:

- Prints its status and the current value of the task code word
- Prints the LIST specification
- Optionally puts the task into a wait state.

If you requested the NOSTOP option, $DEBUG prints task status as "taskname CHECKED AT XXXX." The STOP option generates a "taskname STOPPED AT XXXX" message.

$DEBUG - Debugging Tool *(continued)*

You can modify the LIST and STOP options for all currently defined breakpoints and trace ranges by entering AT ALL. Similarly, you can alter the specifications for the most recently entered AT command with the AT * option.

If you specify LIST UNMAP after issuing the AT command, the unmapped storage area to be listed must already be initialized at the time you set the breakpoint.

Notes:

1. You cannot set breakpoints in ATTNLIST routines.
2. If a trace range is set around a GETVALUE coded with PROMPT=COND and a message data set prompt, the instruction will not wait for input. The input data area will be unchanged.

Syntax:

```
AT ADDR address overlay# NOLIST/LIST NOSTOP/STOP
AT TASK taskname start-add end-add NOLIST/LIST NOSTOP/STOP
AT ALL NOLIST/LIST NOSTOP/STOP
AT *   NOLIST/LIST NOSTOP/STOP
```

Operands	*Description*
ADDR	Keyword indicating this is an instruction program breakpoint specification.
address	Instruction address where you want to insert a breakpoint. **Note:** Be sure that this is the address of the first word of an executable instruction, since $DEBUG will modify this word. $DEBUG can give unpredictable results if you specify the address of data or the address of other than the first word generated by an instruction.
overlay#	Overlay segment number where the system sets the breakpoint when the address you specified is within the overlay area. You can find the overlay segment number in the link map of the program you are debugging.
NOLIST	You need no special print request at this breakpoint or trace range.
LIST	Complete specification for a storage or register display; see the LIST command for a description of all list options and parameters.
NOSTOP	Processing continues after the breakpoint notification occurs.
STOP	The task stops whenever the system satisfies this breakpoint or trace range specification.

$DEBUG - Debugging Tool *(continued)*

TASK	Trace range specification.
taskname	Name of task you want to trace (if the program contains only one task, omit this parameter).
start-add	Trace range starting address (since your program is not actually modified by a trace specification, you don't have to use special care when you enter trace addresses).
end-add	Trace range ending address.
ALL	The system redefines all currently defined breakpoints and trace ranges with new list and stop options.
*	The system redefines the most recently defined breakpoint or trace range specification. The system determines this specification by the last usage of an AT, GO, or OFF commands.

BP — List Breakpoints and Trace Ranges

BP displays all breakpoints and trace ranges that you specified for the current debug session. For each breakpoint, BP displays the task address, the instruction type, the associated list options, and the overlay segment number, and it indicates whether you specified a stop.

Syntax:

```
BP

Required: none
```

Operands ***Description***

None

CLOSE — Close Spool Job Created by $DEBUG

CLOSE closes the spool job created by the last PRINT command you issued to a spoolable device.

Example: CLOSE command.

```
> CLOSE
SPOOL JOB CLOSED
```

$DEBUG - Debugging Tool *(continued)*

A spool job is created when a LIST command is sent to a spooling device. Issueing a CLOSE command will close this spool job and make it ready for printing. Only the spool job associated with the most recent PRINT command is closed. IF $SPOOL is active and there is no job to close (in other words, no LIST command has been issued between a PRINT command and a CLOSE command,) a spool job will be created consisting of an ENQT and a DEQT. If $SPOOL is not active, such a command will be ignored. Delete such empty jobs to release space.

If $SPOOL is not active, you will receive the following message:

```
> CLOSE
SPOOLING NOT ACTIVE
```

Syntax:

```
CLOSE

Required: none
```

Operands ***Description***

none

END — End $DEBUG

END removes all currently-active breakpoints and trace ranges, activates all currently-stopped tasks, and ends $DEBUG. Do not use the $C operator command to cancel $DEBUG.

Note: If the program you are debugging continues to execute after you ended $DEBUG, then you can cancel the program by pressing the attention key and entering the $C operator command and the program name.

Syntax:

```
END

Required: none
```

Operands ***Description***

None

$DEBUG - Debugging Tool *(continued)*

GO — Activate a Stopped Task

GO reactivates any task that $DEBUG has stopped. If a task stops at a breakpoint, specify the exact breakpoint address. If a task stops as a result of a trace specification, supply the name of the task and an address range which brackets the addresses in the original trace request. If you are debugging only one task, you don't need to specify any operands.

Syntax:

```
GO ADDR address
GO TASK taskname
GO ALL
GO *
GO
```

Operands	*Description*
ADDR	Keyword indicating this is a breakpoint specification.
address	Instruction address where the task stops.
TASK	Keyword indicating this is a trace range specification.
taskname	Name of task you want activated. For programs containing only a single task, omit this parameter.
ALL	Activate all currently stopped tasks.
*	Use the most recently referenced breakpoint or trace range specification. The system determines this specification by the last usage of an AT, GO, or OFF command.

$DEBUG - Debugging Tool *(continued)*

GOTO — Change Execution Sequence

GOTO reactivates, at a different instruction, any task that has stopped at an Event Driven Language or Series/1 assembler instruction. If you used a breakpoint or trace to stop the task, supply the current address and the address at which execution should be resumed. $DEBUG will not change the breakpoint or trace specifications.

Syntax:

```
GOTO current-address new-address
```

Operands	***Description***
current-address	Address where the task stops.
new-address	Address where you want execution restarted.

HELP — List $DEBUG Commands

The HELP command produces a list of $DEBUG commands and functions.

Syntax:

```
HELP

Required: none
```

Operands ***Description***

None

$DEBUG - Debugging Tool *(continued)*

LIST — Display Storage or Registers

LIST displays the contents of storage locations, task registers, hardware registers, or the IAR (instruction address register). You can display the LSB (level status block) by listing the IAR with a length of 11 words. Any register data is guaranteed to be current only if $DEBUG stops the corresponding task by a breakpoint or trace range. Use LIST * to repeat the most recently specified LIST command or to verify (list) a patch you have just entered.

The following example shows the LIST command prompts. You can list the unmapped as well as the mapped storage that your program acquired previously with the GETSTG command.

Example: LIST command for unmapped storage.

```
> LIST
OPTION(*/ADDR/R0...R7/#1/#2/IAR/TCODE/UNMAP): UNMAP
STORBLK ADDRESS (0 TO CANCEL LIST): 34
SWAP#: 1
DISPLACEMENT: 12
LENGTH: 50
MODE(X/F/D/A/C): X
```

Syntax:

```
LIST *
LIST ADDR address      length mode
LIST R0...R7  taskname length mode
LIST #1...#2  taskname length mode
LIST IAR      taskname length mode
LIST TCODE    taskname length mode
LIST UNMAP    storblkaddress swap# displacement length mode
```

Operands	*Description*
*	Use the most recently specified LIST or PATCH specification. The system determines this by the last usage of a LIST or PATCH command.
ADDR	Keyword indicating this is a display of a mapped storage location.
R0...R7	One of the Series/1 hardware registers R0 through R7 where you want $DEBUG to start the list.

$DEBUG - Debugging Tool *(continued)*

taskname	Name of task where you want $DEBUG to display register data. For programs containing only a single task, omit this parameter.
#1/#2	Task register #1 or #2 specification.
IAR	Keyword indicating you want $DEBUG to display the IAR (instruction address register).
TCODE	Keyword indicating you want $DEBUG to display the task return code words (first two words of the TCB).
UNMAP	Keyword indicating this is a display of an unmapped storage location.
storblk address	Address of the storage statement that defines the unmapped storage location you want $DEBUG to display. (The address can be found in the program listing as the address of the storblk statement.)
swap#	Number of the unmapped storage area instruction. If you specify 0, you can list mapped storage.
displacement	Number of bytes (in hex) where you want to start listing an unmapped storage area. For example, if you enter 1A, $DEBUG begins patching from byte 26 of the unmapped storage area indicated. The largest possible value for displacement is X'FFFF' since the maximum size of an unmapped storage area is 64K.
length	Length of display in words, doublewords, or characters depending on mode.
mode	Mode of data display:

X	-	hexadecimal word
F	-	decimal number(word)
D	-	decimal number(doubleword)
A	-	relocatable address
C	-	EBCDIC character

$DEBUG - Debugging Tool *(continued)*

OFF — Remove Breakpoints and Trace Ranges

OFF removes a breakpoint or trace range established with the AT command. To remove a breakpoint, specify the exact breakpoint address. To remove a trace request, specify the name of the task and an address range which brackets the addresses in the original trace request. If a task currently is stopped at the requested breakpoint or trace range, the system automatically reactivates the task.

Syntax:

```
OFF ADDR address overlay#
OFF TASK taskname start-add end-add
OFF ALL
OFF *
```

Operands	*Description*
ADDR	Keyword indicating this is the removal of the breakpoint specification.
address	Instruction address where the system previously established a breakpoint.
overlay#	Overlay segment number from where you want $DEBUG to remove the breakpoint when the address specified is within the overlay area. You can find the overlay segment number in the link map of the program you are debugging.
TASK	Keyword indicating you want $DEBUG to remove a trace range.
taskname	Name of task associated with a trace range; for programs containing only a single task, omit this parameter.
start-add	Trace range starting address.
end-add	Trace range ending address.
ALL	Remove all breakpoints and trace ranges.
*	Use the most recently referenced breakpoint or trace range specification. The system determines this by the last usage of an AT, GO, or OFF command.

$DEBUG - Debugging Tool *(continued)*

PATCH — Modify Storage or Registers

PATCH modifies the contents of storage locations, task registers, hardware registers, task code words, and the IAR (instruction address register). You can modify the entire LSB (level status block) by patching the IAR with a length specification of 11 words. The patch to any register data is guaranteed only if a $DEBUG breakpoint or trace range stops the corresponding task. To respecify the data for the most recent patch or to patch the data the most recent LIST command displays, enter PATCH *.

The following example shows the PATCH command prompts. You can patch unmapped as well as mapped storage. Unmapped storage may be patched only after the GETSTG statement is issued.

Example: PATCH command for unmapped storage.

```
> PATCH
OPTION(*/ADDR/R0...R7/#1/#2/IAR/TCODE/UNMAP):  UNMAP
STORBLK ADDRESS (0 TO CANCEL PATCH):  34
SWAP#:  1
DISPLACEMENT:  12
LENGTH:  1
MODE(X/F/D/A/C):  X
NOW IS
 0012 X' 0000'
DATA:  FFFF
NEW DATA
 0012 X' FFFF'
YES/NO/CONTINUE:  Y
```

After you enter the patch command, the system displays the current storage or register content, and you are prompted for the patch data (a string of data entries that satisfies the length and mode specifications). Use spaces to separate the entries. After you enter the patch data, you can apply the patch by responding YES, cancel the patch by responding NO, or indicate additional patches by responding CONTINUE to the prompting message. If you respond CONTINUE, the system performs the patch and prompting continues for new length, mode, and data specifications to storage or register locations immediately behind your previous patch.

If you enter less data than specified with the length operand, the effective patch is padded to the right with blanks for character data and zeros for all other data types.

$DEBUG - Debugging Tool *(continued)*

Syntax:

```
PATCH *
PATCH ADDR address      length mode
PATCH R0...R7  taskname length mode
PATCH #1/#2    taskname length mode
PATCH IAR      taskname length mode
PATCH TCODE    taskname length mode
PATCH UNMAP    storblkaddress swap# displacement length mode
```

Operands	***Description***
*	Use the most recently referenced LIST or PATCH specification. This is determined by the last usage of a LIST or PATCH command.
ADDR	Keyword indicating this is a mapped storage patch.
R0...R7	One of the Series/1 hardware registers R0 through R7, where you want $DEBUG to start the patch.
taskname	Name of task for which you want $DEBUG to modify register data. For programs containing only a single task, omit this parameter.
#1/#2	Task register #1 or #2 specification.
IAR	Keyword indicating you want the IAR (anstruction address register) to be modified.
TCODE	Keyword indicating which task return code word(s) (first two words of the TCB) you want modified.
UNMAP	Keyword indicating this is an unmapped storage patch.
storblk address	Address of the storage statement that defines the unmapped storage location you want $DEBUG to display. (You can obtain this address from the program header on a copy of your application program.)
swap#	Number of the unmapped storage area instruction. If you specify 0, you can list mapped storage.

$DEBUG - Debugging Tool *(continued)*

displacement	Number of bytes (in hex) where you want to start listing an unmapped storage area. For example, if you enter 1A, $DEBUG begins patching from byte 26 of the unmapped storage area indicated. The largest possible value for displacement is X'FFFF' since the maximum size of an unmapped storage area is 64K.
length	Length of patch in words, doublewords, or characters depending on mode.
mode	Mode of data entry: X - hexadecimal word F - decimal number(word) D - decimal number(doubleword) A - relocatable address C - EBCDIC character

POST — Post an Event or Process Interrupt

POST activates a task waiting for an event or a process interrupt. To duplicate a previous posting, enter POST *. The address of the ECB (event control block) that the system will post is contained in the second word of a WAIT instruction as shown on a program assembly listing. You can also post process interrupts by name using the PIxx option.

Note: Be sure to enter a valid ECB address; an invalid address will result in unpredictable results.

Syntax:

```
POST ADDR address code
POST PIxx code
POST *
```

Operands	*Description*
ADDR	The address of an ECB (event control block).
address	ECB address you want posted.
code	Decimal code you want posted to the specified ECB.
PIxx	Name of process interrupt PI1 to PI99.
*	Use the most recently referenced ECB address or PIxx name and code specification.

$DEBUG - Debugging Tool *(continued)*

QUALIFY — Modify Base Address

QUALIFY modifies the base address that $DEBUG uses to refer to physical storage locations.

Syntax:

```
QUALIFY base displ
Q base displ
```

Operands	*Description*
base	New hexadecimal base address.
displ	Hexadecimal offset you want added to the base to form the new base address for all subsequent address references. Enter the origin of the program module as shown on the link editor listing.

PRINT — Direct Output to Another Terminal

PRINT allows you to direct the output to a terminal other than the one you used to invoke $DEBUG.

Syntax:

```
PRINT terminal-name
PRINT *
PRINT
```

Operands	*Description*
terminal-name	The name of the terminal where you want the output directed. To direct the output back to the current terminal, enter a blank or * to indicate the terminal you used to invoke $DEBUG.

$DEBUG - Debugging Tool *(continued)*

WHERE — Display Status of All Tasks

WHERE displays the current status of each task. If a task is currently processing its breakpoint routine, the system marks it CHECKED. If a breakpoint or trace request has stopped a task or if $DEBUG has not yet attached the main task, the system marks the task STOPPED. In all other cases, the system shows that each task is at the currently executing instruction, at the command it will start executing when dispatched by the task supervisor, or at the last command executed prior to entering a wait state.

Note: $DEBUG can only locate tasks within a program if the task control blocks (TCBs) are chained together. This chaining takes place at program assembly time for all tasks that are part of the assembly containing the main program task. Tasks that are assembled separately and then linked to the main task will not have their TCBs chained together until the system ATTACHs the task at program execution time. (See a description of the ATTACH instruction in the *Language Reference*.) For $DEBUG to control tasks that are linked to the main task, you must load into storage the program you are debugging, and you must attach the desired tasks before you load $DEBUG to control the further execution of the tasks.

Syntax:

```
WHERE
WH

Required: none
```

Operands ***Description***

None

$DICOMP - Display/Modify Profiles

Use the $DICOMP utility to add display profiles to the composer and to modify existing display profiles. Because this utility does not change the basic structure of the online data base, you can use it at the same time you are performing other functions. You can use $DIINTR to cause the system to generate a partial display. If you need corrections or additions, use $DICOMP to alter the display profile.

Invoking $DICOMP

You invoke $DICOMP with the $L operator command or option 5.2 of the session manager.

To start execution of $DICOMP:

1. Load the program $DICOMP specifying the appropriate data set name. You can use $DIFILE, the online data set, or any other data set. However, you should make sure that another user or program is not changing or using the same data set.

```
> $L $DICOMP
DISPLIB (NAME,VOLUME):
LOADING $DICOMP     44P,00:19:34, LP= 9200, PART=1
```

2. The system responds with the program-loaded message followed by:

```
$DICOMP - DISPLAY DATA BASE COMPOSER
COMMAND (?):
```

$DICOMP Commands

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

```
COMMAND (?):  ?

AD - ADD A NEW MEMBER
AL - ALTER EXISTING MEMBER
EN - EXIT PROGRAM
IN - INSERT OR DELETE DISPLAY ELEMENTS IN A MEMBER
PR - PRINT MEMBER FORMATTED
TD - TEST DISPLAY

COMMAND (?):
```

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

$DICOMP - Display/Modify Profiles *(continued)*

AD — Add a New Member to Data Base

Use the AD command to generate a new display profile. The display can be either report or graphic form. You are prompted to enter a 1-8 character display profile name that the interpreter will then use to retrieve the display. The next prompt message asks if you want a display heading. If you respond yes, the system assumes you want a report display. If you respond no, the system assumes you want a graphic display and prompts you to proceed with generating the display. You can also select the device where you want the system to route the output from the interpreter.

Report Display: If you respond N to the question

```
IS THIS A GRAPHIC DISPLAY?
```

$DICOMP prompts you to enter the column headings you want. The system allows one line, up to 132 characters. Following your entry of the column headings, $DICOMP prompts you to enter the name of the print report data member and then to enter the next command.

Graphic Display: If you want a graphic display, you should respond Y to the question

```
IS THIS A GRAPHIC DISPLAY?
```

The composer then asks if you want a 3-D object display. If you respond Y, then all following references to X and Y values will also include the Z value. The composer asks you to enter the values X, Y, and Z. The system uses them to position the first character of the display heading. $DICOMP then prompts you for a command, COMMAND (?):. You can use the "Composer Subcommands" on page UT-147 now to add, change, or insert elements in your display.

AL — Alter an Existing Member

Use the AL command to display each element of a display profile and make changes, using subcommands, provided you do not change the size of the element and the sequence of commands. This command is of great value during the trial-and-error period when you are generating a new display. You can generate a display using the AD command and display the results using the interpreter. You are allowed to start alteration at the beginning of the member and display each element in turn or to skip to a specific element within the member. Use the PR command to display the elements and their sequence numbers. As the system displays each element, it questions you whether or not you want to alter this element.

```
ALTER ENTRY?
```

If you choose to alter this element, $DICOMP prompts you to reenter the element as described previously in the AD command. When the system reaches the end of the display profile, the composer ends and you can redisplay the profile to see if you are satisfied with the corrections.

$DICOMP - Display/Modify Profiles *(continued)*

EN — Exit Program

Use the EN command to exit immediately from the composer.

IN — Insert or Delete Elements in an Existing Member

Use the IN command to combine the facilities of the AL and AD commands with the ability to delete individual display elements. Because the IN command creates a new member in the data base, you can change the size and sequence of display elements.

Note: We recommend using the $DICOMP utility to verify that sufficient space remains in the data base. By using the $DIUTIL utility (LA and ST commands), you can determine the size of the member you want to modify and the remaining space in the data base. As described in the AL command, the composer displays each element in turn, asking the following questions:

```
KEEP ENTRY?
DELETE ENTRY?
ALTER ENTRY?
```

If you elect to keep the entry, the composer proceeds to the next element. If you respond N, the system displays the DELETE ENTRY? question. If you respond N again, the system displays the ALTER ENTRY? question. If you respond Y to'this prompt, the composer proceeds with the alteration process as described in the AL command.

Following the alteration of the display, the system returns control to the ID command and repeats the process for the next element. If you did not alter the element, the system prompts you to insert a new subcommand. At this point, all the functions of the AD command are available. You can add one display element. the system then returns control to the ID command and redisplays the previous element and repeats the sequence.

Again, as in the alteration procedure, you must step through each element in the display profile before completion. When the system reaches the end of the display, it issues the following message:

```
END OF DATA - ISSUE SAVE OR ADD NEW COMMANDS
```

The composer then returns to the AD command and you can enter additional commands.

Note: You must issue an SA (save) subcommand to end insertion of data. When you issue the SA subcommand, the composer deletes the old member and renames the newly-built member with the old name. This procedure makes the modified version available to the interpreter. It is recommended that you use $COMPRES to compress the data base following insert activity to prevent fragmentation of the data base and reclaim unused space.

$DICOMP - Display/Modify Profiles *(continued)*

PR — Print Member Formatted

Use the PR command to display, on the terminal or printer, the contents of a display profile member formatted the same way as the AL and IN commands. This display is useful as an aid in maintaining display profiles. To obtain a high-speed hard copy, direct the listing to the $SYSPRTR.

TD — Test Display as Currently Entered

When you issue the TD command, $DICOMP prompts you for the name of a plot control member and then invokes $DIINTR to generate the specified display. The system returns control to you to make changes.

Composer Subcommands

When adding, altering, or inserting elements in a member, use subcommands. These are listed below and described on the following pages. When you enter a subcommand, the system places it in to modify the member. The interpreter can use the member later to generate the desired display. You can use the following subcommands:

```
AD - ADVANCE X,Y
DI - DIRECT OUTPUT
DR - DRAW A SYMBOL
EN - EXIT PROGRAM
EP - END DISPLAY
HX - SEND DATA (HEX)
IM - INSERT MEMBER
JP - JUMP TO ADDRESS
JR - JUMP REFERENCE
LB - DISPLAY CHARACTERS
LI - DRAW A LINE TO X,Y
LR - DRAW LINE RELATIVE
MP - MOVE BEAM TO X,Y
PC - PLOT CURVE ONLY
PL - PLOT DATA
RT - ACTIVATE REALTIME DATA MEMBER
SA - SAVE ACCUMULATED DATA
TD - DISPLAY TIME AND DATE
VA - DISPLAY VARIABLE
```

Method for Producing a Graphic Display

The suggested method to produce a graphic display is to draw the display on graph paper first and assign X and Y coordinates to the key nodes in the display. Then use this drawing as a guide to the generation of the display, keeping in mind the screen limits of the terminal you will use. The view area of the graphic terminals supported is shown in Figure 11 on page UT-148. Figure 12 on page UT-148 shows the space supported in 3-D mode.

$DICOMP - Display/Modify Profiles *(continued)*

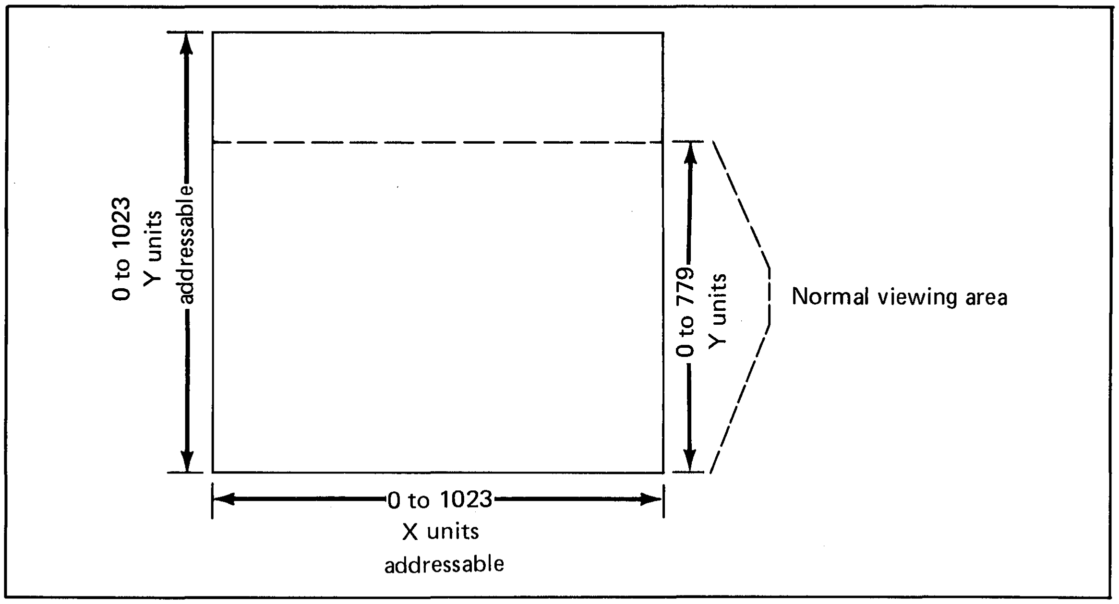


Figure 11. X,Y coordinate grid and viewing area

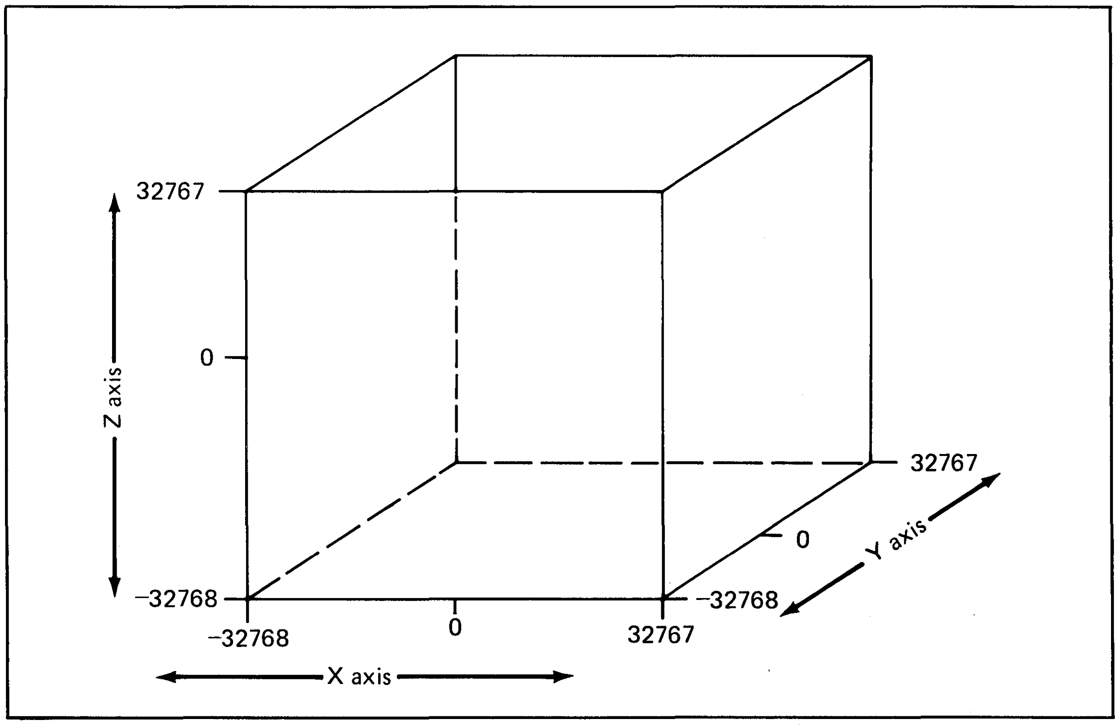


Figure 12. X,Y,Z coordinate grid and viewing area

$DICOMP - Display/Modify Profiles *(continued)*

AD — Advance X,Y

Use the AD subcommand to move the beam position by the value specified. This can be helpful in displaying data with even spacing on the screen. After issuing a DR subcommand using a symbol, AD advances the X,Y position to the next position without regard to the actual screen X,Y location. The limit for the specified X or Y value is plus or minus 512 units. If you are defining a 3-D object, then the system requests the Z axis value as well.

DI — Direct Output

Use the DI subcommand to direct the resulting graphic output to a terminal other than the one you used to enter commands. The terminal name you enter is the label of the TERMINAL statement used to describe the desired terminal.

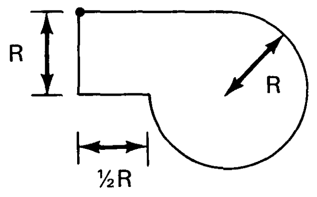
DR — Draw a Symbol

Use the DR subcommand to draw a predefined symbol. Several commonly used symbols have been provided. In specifying a symbol, you are prompted to enter the symbol number and the symbol modifier. These values are used by the interpreter to generate the requested symbol. Some of the symbols require additional information. If so, the system prompts you for this additional information. Valid symbol numbers are 1 through 14. The following examples illustrate specifying symbols 1 through 14.

Symbol # - 1: Draw fan symbol left hand format.

Modifier = Radius of fan body and opening on left side of fan. Must be a multiple of 4.

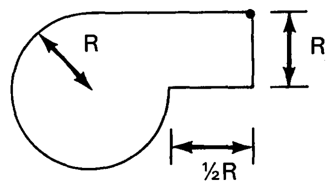
= start and end X,Y current position.



Symbol # - 2: Draw fan symbol right hand format.

Modifier = Radius of fan body and opening on right side of fan. Must be a multiple of 4.

= start and end X,Y current position.

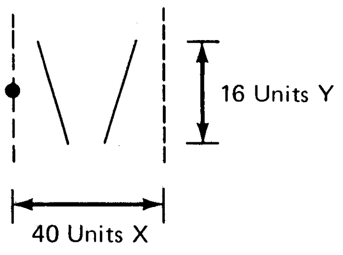


$DICOMP - Display/Modify Profiles *(continued)*

Symbol # - 3: Draw damper vertical.

Modifier = Number of damper pairs
to be generated in the down direction.

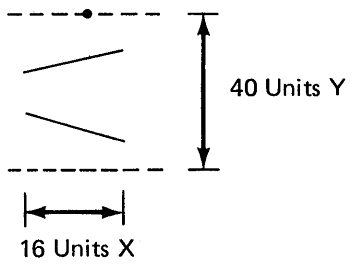
= start and end X,Y current position.



Symbol # - 4: Draw damper horizontal.

Modifier = Number of damper pairs
to be generated to the right.

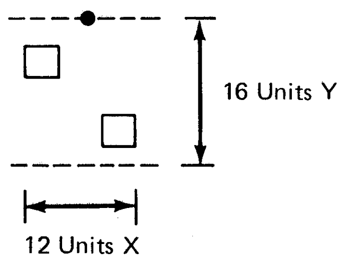
= start and end X,Y current
position.



Symbol # - 5: Draw a hot coil.

Modifier = Number of hot coil pairs
to be generated in the down direction.

= start and end X,Y current
position.

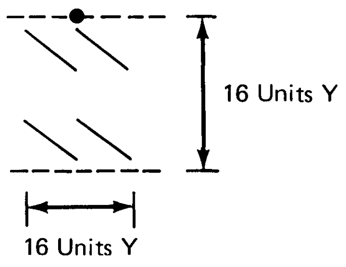


$DICOMP - Display/Modify Profiles *(continued)*

Symbol # - 6: Draw a cold coil.

Modifier = Number of double pairs
to be generated in the down direction.

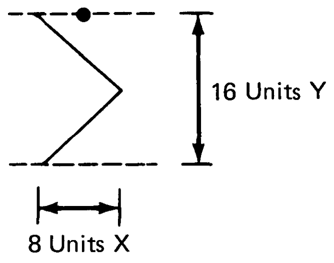
= start and end X,Y current
position.



Symbol # - 7: Draw a filter element.

Modifier = Number of elements
to be generated in the down direction.

= start and end X,Y current position.

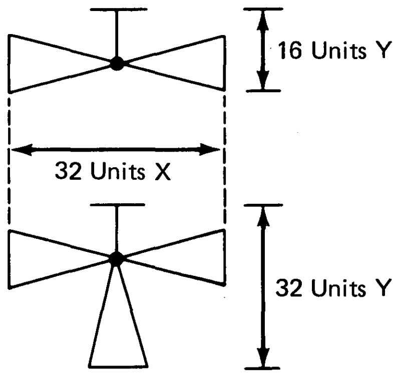


Symbol # - 8: Draw a valve.

For 2-way valve
Modifier = 2

For 3-way valve
Modifier = 3

= start and end X,Y current position.



$DICOMP - Display/Modify Profiles *(continued)*

Symbol # - 9: Draw an arrow.

Modifier =

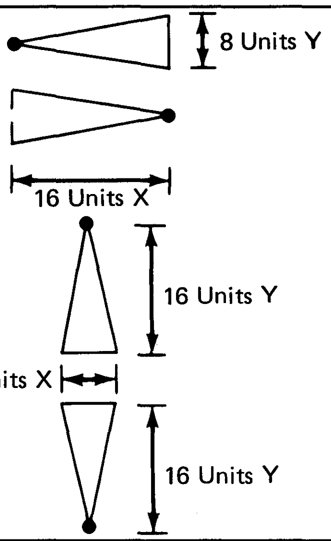
1 for left

2 for right

3 for up

4 for down

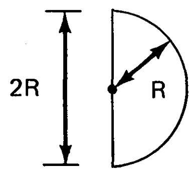
● = start and end X,Y current position.



Symbol # - 10: Draw a logic block right.

Modifier = Radius of half circle.
Must be a multiple of 4.

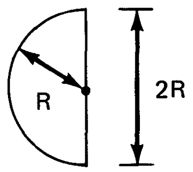
● = start and end X,Y current position.



Symbol # - 11: Draw a logic block left.

Modifier = Radius of half circle.
Must be a multiple of 4.

● = start and end X,Y current position.

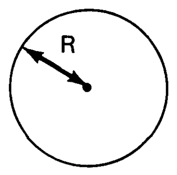


$DICOMP - Display/Modify Profiles *(continued)*

Symbol # - 12: Draw a circle.

Modifier = Radius of Circle.
Must be a multiple of 4.

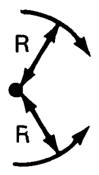
= start and end X,Y current position.



Symbol # - 13: Draw an arc right.

Modifier = Radius of circle.
Must be a multiple of 4.

= start and end X,Y current position.



Note:
This symbol requires additional values.

1. Draw arc up or down. Enter zero for down or one for up.

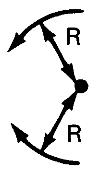
2. Number of Y units to draw arc. Must be a multiple of 4.

Note:
This symbol always starts at X=0 and proceeds until the Y units have been exhausted.

Symbol # - 14: Draw an arc left.

Modifier = Radius of circle.
Must be a multiple of 4.

= start and end. X,Y current position.



Note:
See note under symbol 13 for additional information.

$DICOMP - Display/Modify Profiles *(continued)*

EN — Exit Program

Use the EN subcommand to terminate without updating the display profile data base directory. All data collected up to this point for this member is lost.

EP — End Display

Use the EP subcommand to specify that the end of this section of the display has been reached. Normally, you would follow this command with the SA subcommand. However, this command can be useful if a jump zero/not zero causes the interpreter to take alternate paths. Use the EP subcommand at the end of each of these paths instead of an unconditional jump to a common ending point.

HX — Send Data

Use the HX subcommand to send up to 16 words of data without conversion to the terminal. All bit patterns are valid; therefore, you can send control or special data to the terminal.

IM — Insert Member

Use the IM subcommand to combine display profile members to form one display. IM allows you to conserve disk space, decrease time required to enter display profiles, and standardize display formats. For example, you can build a display profile member to represent a common background of a physical system or floor plan. Then, by defining another display profile member, you can superimpose on the background the variables that will make the display unique. the system permits only one level of nesting. That is, a member you insert using the IM subcommand cannot contain any IM subcommands. However, a primary member can include multiple IM subcommands.

JP — Jump to Address

Use the JP subcommand to change the sequence of execution of subcommands. There are three types of "jump to address" subcommands that you can use. They are:

- Jump Unconditional
- Jump if Zero
- Jump if Not Zero

As described in the display variable command, the conditional jump commands are dependent on the use of the realtime data member. If you select conditional jump, then the jump is based on the current condition (zero/not zero) of the specified word and record. Jump unconditional prompts you to enter a JR subcommand. This reference is two characters and is resolved when you define a JR subcommand (see the JR subcommand definition). If you select a conditional jump, the system issues prompt messages requesting word number and record number. Following the definition of these two codes, the system prompts you to enter the JR subcommand. The jump to reference for a conditional jump is the same as that of an unconditional jump. The following example shows the use of the JP subcommand.

$DICOMP - Display/Modify Profiles *(continued)*

Command sequence using jump:

```
MP   X=200, Y=200
JP   Zero, WORD#=0, RECORD#=4, JR=AA
DR   SYM=1, MOD=40
JP   JR=BB
JR   AA
DR   SYM=2, MOD=40
JR   BB
EP
SA
```

The preceding example draws a fan symbol at 200,200 either right or left depending on the zero/not zero condition of the realtime data member word 0, record 4.

In the preceding sequence, the first JP causes a jump to JR AA if word 0 of record 4 is zero. The second JP causes an unconditional jump to JR BB.

JR — Jump Reference

Use the JR subcommand to indicate to the composer that this location in the command sequence is referred to in a JP subcommand. The location is defined by 2 characters. If you have used these characters already, the system issues an error message. If you exceed the capacity of the JR table, the system issues an error message. The capacity of the jump reference table is 40 unique jump reference points for each display.

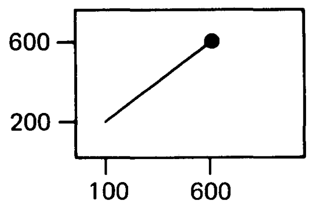
LB — Display Characters

Use the LB subcommand to place a character string on the screen. You do not have to use an MP subcommand to position the beam because LB allows specification of the location of first character. If you are defining a 3-D object, then the system requests X, Y, and Z values. The system can display up to 72 characters. The ending X,Y position is 1 character position beyond the last character in the string.

LI — Draw a Line to X,Y

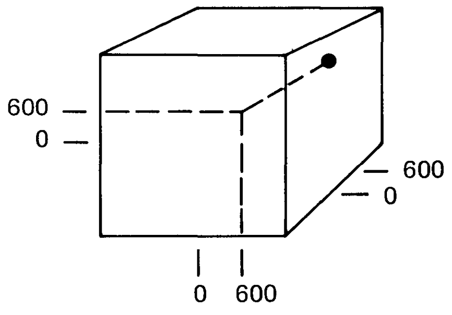
Use the LI subcommand to draw a vector to the specified X and Y coordinates from wherever you left the beam with the previous command.

$DICOMP - Display/Modify Profiles *(continued)*



Draw line to X=600 Y=600 Line shown on screen
END X,Y Current position

When generating a 3-D display, the system requires 3 values. These values are X, Y, and Z.



Draw line to X=600 Y=600 Z=600
END X,Y,Z

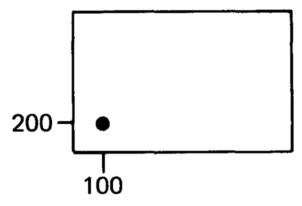
LR — Draw Line Relative

Use the LR subcommand to draw a line relative to the current position. For example, you can (through the use of the MP, JP, and JR subcommands) position the beam at various current positions based on Realtime Data Member conditions. Then you can draw a series of lines to form a symbol using the LR subcommand. This would have the effect of placing the symbol at various screen locations based on external conditions. The limits allowed for the X,Y values are plus or minus 512 units. If you are defining a 3-D object, then the system also requests Z axis value.

$DICOMP - Display/Modify Profiles *(continued)*

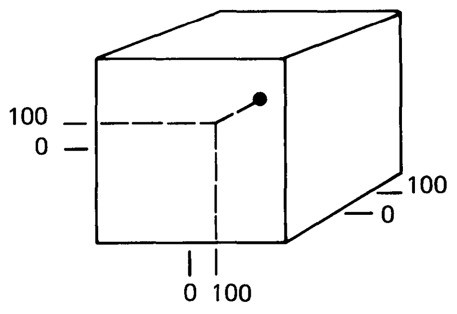
MP — Move Beam to X,Y

Use the MP subcommand to draw a dark vector to the specified X and Y coordinates. A dark vector is not visible and, therefore, results in moving the beam to the specified location.



Beam moved to X=100 Y=200 Nothing shown on screen
END X,Y Current position

When generating a 3-D display, the system requires 3 values. These values are X, Y, and Z.



Beam moved to X=100 Y=100 Z=100
END X,Y,Z Current position

PC — Plot Curve Only

Use the PC subcommand to provide multiple curves on an existing background as defined by a preceding PL command. Refer to the following section (PL) for descriptions of entry procedure. Steps 9 and 10 in that section are the only required actions. You can include as many PC subcommands as you need to obtain the desired results

PL — Plot Data

Use the PL subcommand to format the viewing area into a basic plotter. The system provides options for X and Y labels as well as X and Y grids. The system prompts you to include the name of a plot curve data member. Refer to "$DIUTIL - Maintain Partitioned Data Base" on page UT-220 for information regarding the allocation and formatting of the plot curve data member. The following illustrates the information that PL requires to format the viewing area into a basic plotter. 1) Enter the number of the Y axis divisions

$DICOMP - Display/Modify Profiles *(continued)*

p.To present a readable display, it is suggested that you make this value under 20. However, if you bypass Y axis division values (Step 7), then you may use larger values. Y axis divisions become unreadable when this value exceeds 125. 2) Enter the number of the X axis divisions

To present a readable display, it is suggested that you make this value under 40. However, if you bypass X axis division values (Step 8), then you may use larger values. X axis divisions become unreadable when this value exceeds 200. 3) Vertical Grid?

A Y answer causes the Composer to include commands to connect the X axis divisions (specified in 2 preceding) to the top of the viewing area. Specifying an N bypasses this feature. 4) Horizontal Grid?

Specifying a Y causes the Composer to include commands to connect the Y axis divisions (specified in 1 preceding) to the right side of the viewing area. Specifying an N bypasses this feature. 5) Enter Y axis label - 24 characters

You must enter the Y axis label. If you do not want an axis label, press the enter key. This label is general in nature and is placed at the left side of the viewing area. This label is vertical, that is, one character appears under the next. 6) Enter X axis label - 24 characters

You must enter the X axis label. If you do not want an X axis label, press the enter key. This label is general in nature and is placed near the lower portion of the plot viewing area. 7) Y axis division values?

If you want Y axis division values, respond with a Y. The composer asks for as many values as you have specified divisions plus 1 (see Step 1). You must enter 6 characters for each division. The first value the system requests is the value for the Y base line and each succeeding value is for the next division in the plus Y direction. 8) X axis division values?

If you want the X axis division values displayed, respond with a Y. The composer asks for as many values as you have specified divisions plus 1 (see Step 2). You must enter 6 characters for each division. The first value the system requests is the value for the X base line and each succeeding value is for the next division in the plus X direction. 9) Enter Name of Member for Plot Data

Enter the name of a plot curve data member. You must must have allocated and initialized this member with the the utility program $DIUTIL. Refer to "$DIUTIL - Maintain Partitioned Data Base" on page UT-220 for procedures on allocating and initializing this member. 10) Is This Plot a Point Plot?

The composer allows you to use of any valid printable character for the plot. If you specify Y, the system requests the plot character you want. If you specify N, then the system uses a normal line for the curve

The preceding steps generate the necessary commands to cause the system to display a basic plot background and superimpose one curve on that background. If you want additional curves, then you must issues PC subcommands next.

$DICOMP - Display/Modify Profiles *(continued)*

RT — Activate New Realtime Data Member

Use the RT subcommand to define multiple realtime data members. This subcommand allows you to switch from one member to another during the generation of a display. The default name for the realtime data member is REALTIME.

SA — Save Accumulated Data

Use the SA subcommand to specify that completion of a display profile has been reached. The composer enters the member name into the directory of the display profile data base and makes it available for the interpreter.

TD — Display Time and Date

Use the TD subcommand to display the current time of day and date from the realtime clocks used by the Event Driven Executive. You are reminded that prior to issuing a TD subcommand, you may have to issue an MP subcommand to position the beam to the display location you want. The TD subcommand displays the time and date in the following format:

```
HH:MM:SS   MM/DD/YY
```

Where: HH is Hours

MM is Minutes

SS is Seconds

MM is Month

DD is Day

YY is Year

VA — Display Variable

Use the VA subcommand to place a data variable from the Realtime Data Member on the screen. $DICOMP issues a prompt message asking if you wish to locate the data at a location other than the current X,Y position. If you are defining a 3-D object, then the system requests X, Y, and Z This subcommand requires that you allocate the realtime data member. The composer continues by asking you for the record number and word number. The record number is the record number within the realtime data member. The word number is the word number within the record specified. This value is in the range of 0–8.

The system requests the function code next and indicates the type of variable to be displayed. Valid function codes are as follows:

0	Single-precision integer

$DICOMP - Display/Modify Profiles *(continued)*

1	Double-precision integer
2	Standard-precision floating point
3	Extended-precision floating point
15	Character data

The system requests type code next. It is an indicator of the format of the value to be displayed. Valid type codes are:

0	Integer
1	Floating-point F format
2	Floating-point E format

The system requests field width and number of decimal places next. If the variable is an integer, the number of decimals should be zero.

$DIINTR - Graphics Interpreter Utility

The $DIINTR interpreter utility searches the data base and generates the display you request. You can generate both graphic and report displays in this manner. Each display profile is made up of many display profile elements. Each element, when retrieved from the data base by the interpreter, is decoded and converted to the appropriate command to cause the system to perform the action you request. Each display profile element contains various parts, such as display code, X and Y coordinates, symbol ID, and symbol modifier. Realtime data member record number and additional member names are included in the display profile element.

Invoking $DIINTR

You invoke $DIINTR with the $L operator command or option 5.3 of the session manager.

To begin operation of the interpreter, you must first load $DIINTR. The system directs output to the terminal that requests the display or as directed by the display profile. Use the following steps to initiate the processor monitor: 1) Load $DIINTR.

```
 > $L $DIINTR
DISPLIB (NAME,VOLUME):
LOADING $DIINTR     38P,00:12:58, LP= 9200, PART=1

$DIINTR - DISPLAY DATA BASE UTILITY
```

2) The system responds with the prompt message:

```
ENTER DISPLAY ID--XXXXXXXX OR EXIT TO TERMINATE
```

3) To terminate the interpreter, enter EXIT. To cause the interpreter to prepare the display, enter the display ID.

Using $DIINTR from an Application Program

You can $DIINTR from an application program to allow displays without operator assistance. Following is an example of loading $DIINTR from an application program:

$DIINTR - Graphics Interpreter Utility *(continued)*

```
    .
    .
    .
*   Your program
    .
    .
    LOAD     $DIINTR,MBRNME,DS=($DIFILE),EVENT=#WAIT,   C
             LOGMSG=NO
    WAIT     #WAIT
    .
    .
MBRNME   DATA   CL8'DISPLAY'
S1       DATA   F'0'       THESE 8 VALUES ARE FOR 3D OBJECTS
S2       DATA   F'0'       *
S3       DATA   F'0'       *
D        DATA   F'0'       *
T        DATA   F'0'       *
R        DATA   F'0'       *
D1       DATA   F'0'       *
T1       DATA   F'0'       *
```

You must supply eight values to describe the manner in which you want the system to display a three-dimensional (3-D) object. Coding of these values is shown in the above example starting with S1 and continuing to T1. The following describes the meaning of these values when you pass them to $DIINTR.

S1	Platform Location X=
S2	Platform Location Y=
S3	Platform Location Z=
D	Platform Direction in Degrees
T	Platform Tilt in Degrees
R	Platform Rotate in Degrees
D1	View Direction in Degrees
T1	View Tilt in Degrees

These values are single-precision integers and may contain a numeric value from -32768 to +32767.

You must have a 4955 processor with floating-point hardware installed to display 3-D images.

Three-dimensional (3-D) Concepts as Used by $DIINTR

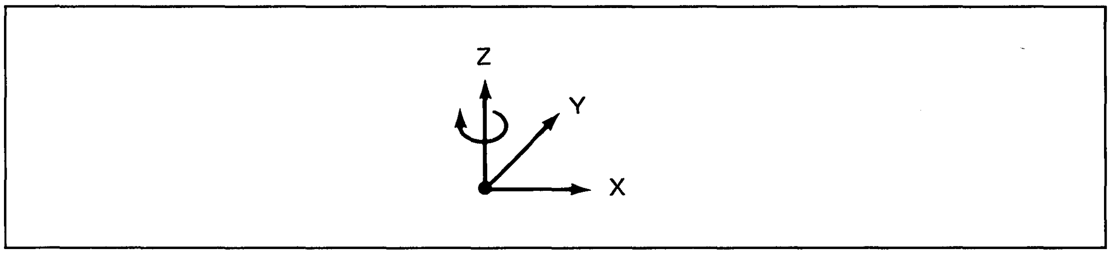
Three-dimensional (3-D) objects can be defined by $DICOMP and placed on disk or diskette in much the same way as with a two-dimensional (2-D) object. The only difference is that each point in space has three values associated with it instead of two. These three values represent the X, Y, and Z coordinates of the point in space. The following illustration shows the limits of the defined area in space. The maximum limits of the defined areas in space are -32768 to +32767. You can define one or more objects within this cube. Once you define the object, you can view it from any location within the same space. To specify the location from where you

$DIINTR - Graphics Interpreter Utility *(continued)*

wish to view the object, either pass these eight values through the use of the PARM= parameter in the LOAD instruction or, if you invoked it by the $L command, wait for $DIINTR to request this input. The concept used to compute the 2-D representation of a 3-D object is as follows. The system assumes the viewer is suspended on a platform at a specific location in space. The first three values are the X, Y, and Z values that define the location in space of the viewing platform. The next five values represent the physical orientation of the platform and the viewer's orientation on that platform.

Platform Direction in Degrees

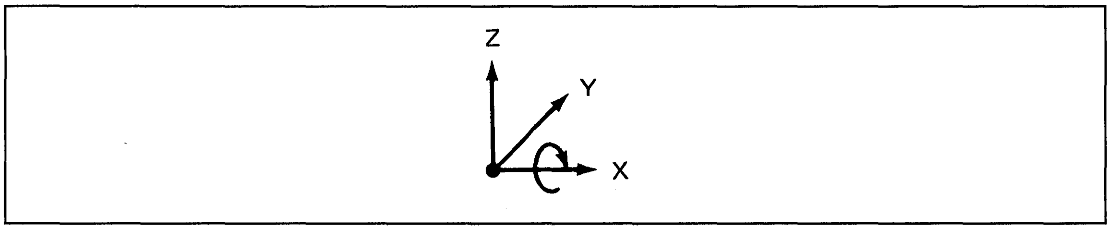
Assume the following unit vector:



If this unit vector is rotated in the direction Y to X around the Z axis, you can turn the view in any direction. A plus value causes the unit vector to rotate clockwise as viewed from the Z axis to zero.

Platform Tilt in Degrees

Assume the following unit vector:

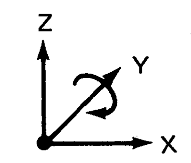


If this unit vector is rotated in the direction Z to Y around the X axis, you can tilt the view to any angle. A plus value causes the unit vector to rotate clockwise as viewed from the X axis to zero.

$DIINTR - Graphics Interpreter Utility *(continued)*

Platform Rotate in Degrees

Assume the following unit vectors:



If this unit vector is rotated in the direction Z to X around the Y axis, you can rotate the view to any angle. A plus value causes the unit vector to rotate clockwise as viewed from the -Y axis to zero.

View Direction In Degrees

The system uses this value in the same way it uses the Platform Direction, but it calculates the value after it computes the above three. This calculation rotates the unit vector in a Y to X direction around the Z axis with a plus value causing the unit vector to rotate clockwise as viewed from the Z axis to zero.

View Tilt In Degrees

The system uses this value in the same way it uses the Platform Tilt, but it calculates the value after it computes the above four. This calculation rotates the unit vector in a Z to X direction around the Y axis with a plus value causing the unit to rotate clockwise as viewed from the -Y axis to zero.

Once the eight values you provided are computed, the system converts the object in space to its 2-D representation and sends it to the terminal. It is possible to view an object with all or a portion of it outside the viewing area. The system does not show points and lines that do not fall within the viewing area. Figure 13 on page UT-165 shows the viewing area.

$DIINTR - Graphics Interpreter Utility *(continued)*

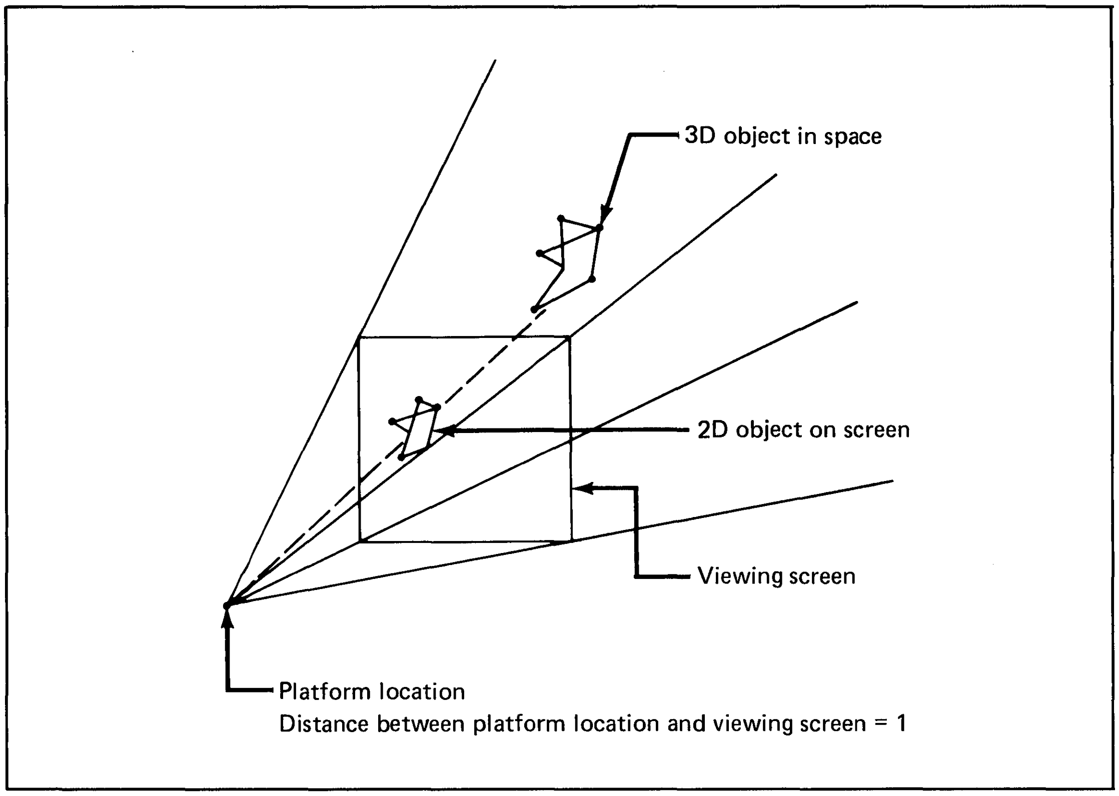


Figure 13. Viewing Area in 3-D Mode

$DIINTR - Graphics Interpreter Utility *(continued)*

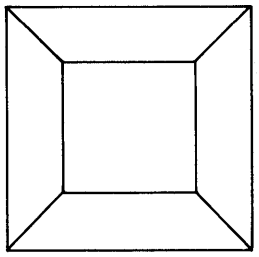
The following example defines a 3-D object in space:

A Cube

CMD	X	Y	Z
MP	-100	-100	-100
LI	+100	-100	-100
LI	+100	-100	+100
LI	-100	-100	+100
LI	-100	-100	-100
LI	-100	+100	-100
LI	+100	+100	-100
LI	+100	+100	+100
LI	-100	+100	+100
LI	-100	+100	-100
MP	+100	-100	-100
LI	+100	+100	-100
MP	+100	-100	+100
LI	+100	+100	+100
MP	-100	-100	+100
LI	-100	+100	+100
EP			
SA			

Object as viewed from:

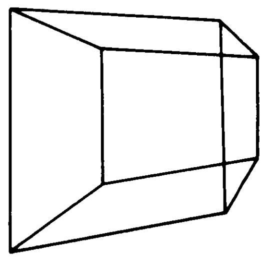
S1	0
S2	-400
S3	0
D	0
T	0
R	0
D1	0
T1	0



$DIINTR - Graphics Interpreter Utility *(continued)*

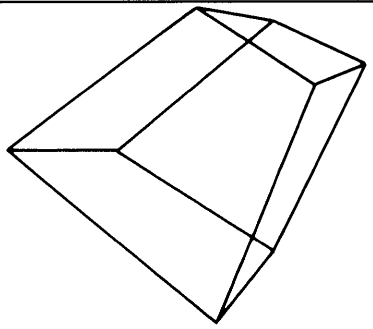
Object as viewed from:

S1	-150
S2	-400
S3	0
D	0
T	0
R	0
D1	0
T1	0



Object as viewed from:

S1	-150
S2	-400
S3	100
D	0
T	0
R	45
D1	0
T1	0



$DIRECT - Directory Organization Sort

$DIRECT sorts a disk or diskette volume directory. $DIRECT sorts alphabetically, by size, by location on disk, or in specific order.

Notes:

1. Allocation or deletion of a data set alters the ordering of the data sets.
2. It is neither possible nor necessary to use the $C command with $DIRECT since the utility patches itself.

Invoking $DIRECT

You invoke $DIRECT with the $L command. When you load it, $DIRECT prompts you to set the terminal to roll screen mode. If you respond Y, it places the terminal in roll screen mode which means you do not need to press the enter key each time the screen fills up. Output "rolls" off the top of the screen as new terminal output appears at the bottom of the screen. If you respond N, you must press the enter key each time the screen fills up.

```
> $L $DIRECT
LOADING $DIRECT     40P,00:23:30, LP= 9200, PART=1

$DIRECT - DIRECTORY SORT
WARNING: VOLUME DIRECTORY WILL
BE ALTERED.  SHOULD ONLY BE RUN
WHEN NO OTHER PROGRAMS ARE ACTIVE!

CONTINUE (Y/N)? Y

SET TERMINAL IN ROLL MODE (Y/N)?
```

$DIRECT then prompts you for the volume you want accessed for the directory sort.

```
VOLUME LABEL: EDX003

COMMAND (?):
```

If you enter an incorrect or nonexistent volume name, $DIRECT issues the following message:

```
VOLUME NOT MOUNTED
RETRY?
```

If you reply Y, $DIRECT prompts you for another volume name. If you respond N, $DIRECT ends.

$DIRECT - Directory Organization Sort *(continued)*

$DIRECT Commands

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

```
COMMAND (?):  ?
  CV ---- CHANGE VOLUME
  LA ---- LIST ALL MEMBERS IN VOLUME
  UT ---- SORT BY - USER INPUT FROM TERMINAL
  UD ---- SORT BY - PREDEFINED DATA SET AS INPUT
  AO ---- SORT BY - ALPHABETICAL ORDER
  DL ---- SORT BY - LOCATION ON DISK(ETTE)
  SA ---- SORT BY - SIZE ASCENDING
  SD ---- SORT BY - SIZE DESCENDING
  EN ---- END PROGRAM
COMMAND (?):
```

After $DIRECT displays the commands, it prompts you with COMMAND (?): again. Then you can respond with the command of your choice (for example, LA). $DIRECT prompts you for any parameters the requested function requires.

Each command and its explanation is presented in alphabetical order on the following pages.

AO — Alphabetical Order Sort

Use the AO command to sort the directory in alphabetical order. This command makes it easier for you to find a data set in a directory list.

Example: Sort alphabetically.

```
COMMAND (?):  AO
ALPHABETICAL SORT USING VOLUME EDX002, CONTINUE (Y/N)?  Y

DIRECTORY SORTED

COMMAND (?):
```

After executing the AO command, the directory looks as follows:

```
NAME        TYPE    FIRST REC      SIZE
$EDXDEF     DATA           82        44
DIRECT      PGM          1294        44
RUNCALC     PGM            77         5
SORTLIST    DATA          801        11
TESTSORT    DATA         1177        83
```

$DIRECT - Directory Organization Sort *(continued)*

CV — Change Volume to be Accessed for Directory Sort

Use the CV command to change the volume you want to access for your directory sort. This command displays the volume the system is using currently and prompts you for the name of the the volume you want to access.

Example: Change volume.

```
COMMAND (?): CV

USING VOLUME EDX003

NEW VOLUME LABEL =EDX002

USING VOLUME EDX002

COMMAND (?):
```

DL — Sort By Location on Disk/Diskette

Use the DL command to sort the directory by the data set location on disk/diskette.

Example: Sort by location.

```
COMMAND (?): DL
SORT BY LOCATION ON DISK(ETTE) USING VOLUME EDX002, CONTINUE (Y/N)? Y

DIRECTORY SORTED

COMMAND (?):
```

After executing the DL command, the directory looks as follows:

```
NAME      TYPE    FIRST REC    SIZE
RUNCALC   PGM            77       5
$EDXDEF   DATA           82      44
SORTLIST  DATA          801      11
TESTSORT  DATA         1177      83
DIRECT    PGM          1294      44
```

$DIRECT - Directory Organization Sort *(continued)*

EN — End $DIRECT

Use the EN command to end the $DIRECT utility.

Example: End $DIRECT.

```
COMMAND (?): EN
$DIRECT ENDED AT 08:36:02
```

$DIRECT - Directory Organization Sort *(continued)*

LA — List All Data Sets in a Volume

Use the LA command to list all the data sets contained in a specified volume. Press the attention key and enter the CA command to cancel the list and return to the COMMAND (?): prompt.

Example: List data sets on EDX002.

```
COMMAND (?):  LA

USING VOLUME EDX002

NAME        TYPE    FIRST REC    SIZE
TESTSORT    DATA         1177      83
RUNCALC     PGM            77       5
DIRECT      PGM          1294      44
SORTLIST    DATA          801      11
$EDXDEF     DATA           82      44

COMMAND (?):
```

SA — Sort By Ascending Data Set Size

Use the SA command to sort the directory by ascending (smallest-to-largest) data set size.

Example: Sort directory in ascending order.

```
COMMAND (?):  SA
SORT BY ASCENDING SIZE USING VOLUME EDX002, CONTINUE (Y/N)? Y

DIRECTORY SORTED

COMMAND (?):
```

After executing the SA command, the directory looks as follows:

```
NAME        TYPE    FIRST REC    SIZE
$EDXDEF     DATA           82      44
RUNCALC     PGM            77       5
SORTLIST    DATA          801      11
DIRECT      PGM          1294      44
TESTSORT    DATA         1177      83
```

$DIRECT - Directory Organization Sort *(continued)*

SD — Sort By Descending Data Set Size

Use the SA command to sort the directory by descending (largest-to-smallest) data set size.

Example: Sort directory in descending order.

```
COMMAND (?):  SD
SORT BY DESCENDING SIZE USING VOLUME EDX002, CONTINUE (Y/N)? Y

DIRECTORY SORTED

COMMAND (?):
```

After executing the SD command, the directory looks as follows:

```
NAME        TYPE    FIRST REC     SIZE
TESTSORT    DATA         1177       83
$EDXDEF     DATA           82       44
DIRECT      PGM          1294       44
SORTLIST    DATA          801       11
RUNCALC     PGM            77        5
```

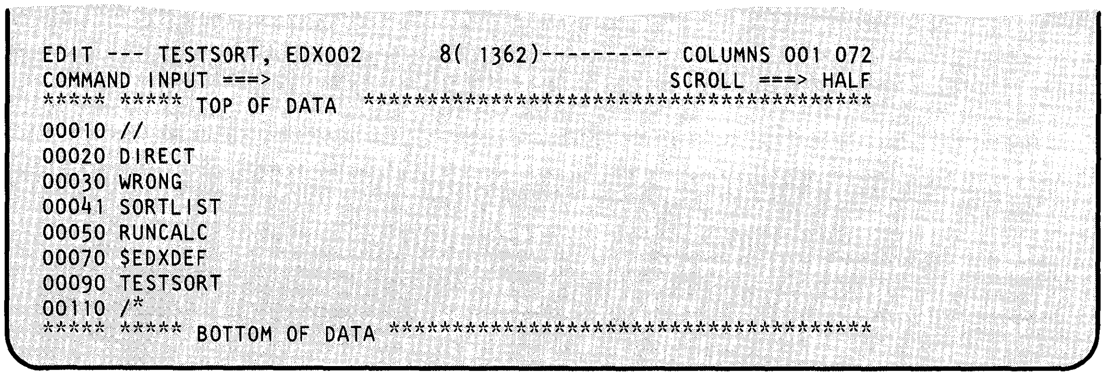
Note: Sorting the directory in descending order can be beneficial if you want to copy the volume to another volume. By placing the largest members at the top of the directory, the system copies them first. This decreases fragmentation of disk space and gives you the best chance of doing the copy without having to compress the target volume.

$DIRECT - Directory Organization Sort *(continued)*

UD — Sort Directory in Predefined Order

Use the UD command to place members in the order you feel is most desirable for retrieval. You can put the most frequently accessed data sets at the top of the directory to increase speed of retrieval. This command prompts you for a previously allocated data set containing the order, by data set name, in which you want the directory sorted. You create this data set using $FSEDIT. The system allows only one data set name for each 80-byte record and you must begin that name in column 1. The first record must be // and the last must be /*. The /* marks the logical end of data which may or may not be the physical end of data.

Example: The following is an example of a data set named TESTSORT on EDX002.



```
EDIT --- TESTSORT, EDX002      8( 1362)---------- COLUMNS 001 072
COMMAND INPUT ===>                               SCROLL ===> HALF
***** ***** TOP OF DATA  ****************************************
00010 //
00020 DIRECT
00030 WRONG
00041 SORTLIST
00050 RUNCALC
00070 $EDXDEF
00090 TESTSORT
00110 /*
***** ***** BOTTOM OF DATA **************************************
```

$DIRECT - Directory Organization Sort *(continued)*

The utility reorders the directory as specified by the input data set.

```
COMMAND (?):  UD
USING VOLUME EDX002

A PREVIOUSLY ALLOCATED DATA SET IS REQUIRED, CONTINUE? Y
ENTER (NAME,VOLUME):  TESTSORT,EDX002

DIRECT HAS BEEN POSITIONED

WRONG  NOT FOUND

SORTLIST HAS BEEN POSITIONED

RUNCALC HAS BEEN POSITIONED

$EDXDEF HAS BEEN POSITIONED

TESTSORT HAS BEEN POSITIONED

DIRECTORY SORTED

COMMAND (?):
```

The utility reorders the directory as follows:

```
NAME        TYPE    FIRST REC      SIZE
DIRECT      PGM          1294        44
SORTLIST    DATA          801        11
RUNCALC     PGM            77         5
$EDXDEF     DATA           82        44
TESTSORT    DATA         1177        83
```

$DIRECT - Directory Organization Sort *(continued)*

UT — Sort Directory in Desired Order Interactively

Use the UT command to place members in the order you feel is most desirable for retrieval. You can put the data sets you access most frequently at the top of the directory to increase speed of retrieval. This command prompts you for member names you want to place at the top of the directory. A blank ends the command.

Example:

```
COMMAND (?):  UT
USING VOLUME EDX002

INTERACTIVE MODE REQUIRES USER INPUT, CONTINUE (Y/N)? Y

ENTER BLANK TO TERMINATE

DATASET #1:  SORTLIST

SORTLIST HAS BEEN POSITIONED


DATASET #2:  DIRECT

DIRECT HAS BEEN POSITIONED


DIRECTORY REORDERED

COMMAND (?):
```

The utility reorders the directory to look as follows:

```
NAME         TYPE    FIRST REC      SIZE
SORTLIST     DATA          801        11
DIRECT       PGM          1294        44
TESTSORT     DATA         1177        83
RUNCALC      PGM            77         5
$EDXDEF      DATA           82        44
```

$DISKUT1 - Allocate/Delete/List Directory Data

$DISKUT1 performs several commonly-used disk or diskette storage management functions. With this utility, you can:

- Rename a data set.
- List data sets.
- Direct listings to $SYSPRTR or terminal.

  Note: For tape management functions, see "$TAPEUT1 - Tape Management" on page UT-522.

Loading $DISKUT1

You load $DISKUT1 with the $L command or option 3.1 of the session manager.

```
> $L $DISKUT1
LOADING $DISKUT1    52P,00:28:08, LP= 9200, PART=1

$DISKUT1 - DATA SET MANAGEMENT, UTILITY I
USING VOLUME EDX002

COMMAND (?):
```

When you load $DISKUT1, it issues the following message:

```
USING VOLUME XXXXXX
```

where XXXXXX equals the IPL volume. The $DISKUT1 commands, with the exception of CV and EN, act upon the IPL volume.

To point to another volume, enter the CV command and the name of the volume. All commands act upon the specified volume until you change them by another CV command or until you end and reload $DISKUT1. If you specify an invalid volume on a CV command, $DISKUT1 uses the IPL volume if it is available. If the IPL volume is not available, the system issues the message NO VOLUME AVAILABLE. You can either end $DISKUT1 or do a CV command with a valid volume.

$DISKUT1 - Allocate/Delete/List Directory Data *(continued)*

$DISKUT1 Commands

To display the $DISKUT1 commands at your terminal, enter a question mark in response to the prompting message COMMAND (?):.

```
COMMAND (?):  ?
AL        --  ALLOCATE A DATA SET
CV        --  CHANGE VOLUME
DE        --  DELETE A DATA SET
DG *      --  DELETE ALL MEMBERS STARTING WITH TEXT
DGP *     --  DELETE ALL PROGRAMS STARTING WITH TEXT
DGD *     --  DELETE ALL DATA SETS STARTING WITH TEXT
DNG *     --  DELETE ALL MEMBERS NOT STARTING WITH TEXT
DNGP *    --  DELETE ALL PROGRAMS NOT STARTING WITH TEXT
DNGD *    --  DELETE ALL DATA SETS NOT STARTING WITH TEXT
SQ        --  SET PROMPT MODES
SNQ       --  RESET PROMPT MODES
EN        --  END THE PROGRAM
LA *      --  LIST ALL DATA SETS
LACTS *   --  LIKE LA BUT IN CTS/RBA MODE
LD *      --  LIST DATA TYPE DATA SETS
LDCTS *   --  LIKE LD BUT LIST IN CTS/RBA MODE
LM        --  LIST A SPECIFIC DATA SET
LP *      --  LIST PROGRAMS
LPCTS *   --  LIKE LP BUT IN CTS/RBA MODE
LS        --  LIST FREE SPACE
LAV *     --  LIST ALL VOLUMES
LAD *     --  LIST DATA SETS ON ALL VOLUMES
LISTP     --  DIRECT LISTING TO $SYSPTR
LISTT     --  DIRECT LISTING TO TERMINAL
RE        --  RENAME A DATA SET
SE        --  SET END OF DATA POINTER/FLAG
          *-  PREFIX (OPTIONAL)

COMMAND (?):
```

$DISKUT1 - Allocate/Delete/List Directory Data *(continued)*

After $DISKUT1 displays the commands, it prompts you once again with the prompt, COMMAND (?):. Then you can respond with the command of your choice (for example, AL). Each command and its explanation is presented in alphabetical order on the following pages.

Notes:

1. You can enter a prefix on the commands that have an asterisk. The prefix can be up to eight (8) characters. If you do specify a prefix, the system lists only those data sets beginning with the prefix.

2. To cancel a long list, press the attention key, press the enter key, then enter CA.

   Note: For a 3101 display terminal, press the attention key and enter CA.

3. If your system includes timer support and you direct output to the $SYSPRTR, the system includes the time and date in the listing.

4. For the 4962 disk and the 4963 disk subsystem, the system shows disk locations in cylinder, track, and sector (CTS) format instead of by record number.

5. For the 4967 disk subsystem, as well as DDSK-30 and DDSK-60 disks, the system shows disk locations in relative block address (RBA) format instead of by record number.

$DISKUT1 - Allocate/Delete/List Directory Data *(continued)*

AL — Allocate a Data Set

Use the AL command to allocate a data set. $DISKUT1 prompts you for the following information:

- The name of the data set
- The size of the data set in records
- The organization type.

The Event Driven Executive recognizes two types of data sets: data-type and program-type. A data-type data set contains work files, user source modules, and application data sets. A program-type data set contains executable (loadable) EDL programs.

Select one of the following organization types:

D	Data organization for data sets used as work files, user source modules, and application data sets.
P	Program organization for data sets that will contain executable (loadable) Event Driven Executive Language programs. Use this for executable object programs (the output of $UPDATE/$UPDATEH).

Example: Allocate a 100-record data-type data set named DATAFILE.

```
COMMAND (?):  AL
MEMBER NAME:  DATAFILE
HOW MANY RECORDS?  100
DEFAULT TYPE = DATA - OK?  Y
DATAFILE CREATED

COMMAND (?):
```

$DISKUT1 - Allocate/Delete/List Directory Data *(continued)*

CV — Change Volume

Use the CV command to change the volume you want to access with other commands. $DISKUT1 prompts you for the new volume label after you enter the CV command.

Example: Change volume.

```
COMMAND (?): CV
NEW VOLUME LABEL =EDX002

USING VOLUME EDX002

COMMAND (?):
```

DE — Delete a Data Set

Use the DE command to delete a data set. $DISKUT1 prompts you for the name of the data set (member) you want to delete.

Example: Delete a data set named DATAFILE

```
COMMAND (?): DE
MEMBER NAME: DATAFILE
DATAFILE DELETE? Y
DATAFILE DELETED

COMMAND (?):
```

$DISKUT1 - Allocate/Delete/List Directory Data *(continued)*

DG — Delete All Members Starting with Text

Use DG to delete data sets that start with a specific prefix. The system displays each data set starting with the specified prefix and $DISKUT1 prompts you as shown in the example. If you do not want to display each data set, use the SNQ command to turn off the prompt mode. $DISKUT1 then deletes the appropriate data sets without verification.

Example: Delete all data sets starting with the prefix $Z.

```
COMMAND (?):  DG
ENTER GENERIC TEXT:  $Z

DELETE ALL MEMBERS STARTING WITH $Z?  Y

CONFIRM DELETE GENERIC REQUEST ON VOLUME:  JMMOBJ
CONTINUE (Y/N)? Y

USING VOLUME JMMOBJ
 NAME     TYPE     FIRST RECORD    SIZE   EOD/PGMSZ

$Z1      DATA          1681         2         NA
 DELETE? Y
$Z1      DELETED
$Z3      DATA          1685         2         NA
 DELETE? Y
$Z3      DELETED
$Z2      DATA          1683         2         NA
 DELETE? Y
$Z2      DELETED

            29 FREE RECORDS IN LIBRARY

COMMAND (?):
```

Respond Y to the DELETE? prompt to delete a data set or N to cancel the delete function. $DISKUT1 continues prompting for each data set on the volume with the specified prefix.

Note: NA means that the end-of-data pointer and flag in the directory member entry have not been set.

DGD — Delete all Data-Type Data Sets Starting with Text

Use the DGP command to delete all data-type data sets starting with a specific prefix. DGD operates in the same manner as DG except that you can only delete data-type data sets.

DGP — Delete All Programs Starting with Text

Use the DGP command to delete all program-type data sets starting with a specific prefix. DGP operates in the same manner as DG except that you can only delete program-type data sets.

$DISKUT1 - Allocate/Delete/List Directory Data *(continued)*

DNG — Delete All Data Sets Not Starting with Text

Use the DNG command to delete all data sets (data- and program-type) not starting with a specific prefix. $DISKUT1 displays each data set that doesn't start with the specified prefix, then prompts you as shown in the example. If you do not want to be prompted for each data set, use the SNQ command to turn off the prompt mode. $DISKUT1 then deletes the appropriate data sets without verifying them.

Example: Delete all data sets that do not start with the prefix $Z.

```
COMMAND (?):  DNG
ENTER GENERIC TEXT:  $Z

DELETE ALL MEMBERS NOT STARTING WITH $Z (Y/N)?  Y

CONFIRM DELETE GENERIC REQUEST ON VOLUME:  JMMOBJ
CONTINUE (Y/N)? Y

USING VOLUME JMMOBJ
 NAME    TYPE     FIRST RECORD    SIZE   EOD/PGMSZ

DATA1    DATA          1681         2        NA
 DELETE (Y/N)?  Y
DATA1    DELETED
MYDATA   DATA          1685         2        NA
 DELETE (Y/N)?  Y
MYDATA   DELETED
DATA2    DATA          1683         2        NA
 DELETE (Y/N)? Y
DATA2    DELETED

            29 FREE RECORDS IN LIBRARY

COMMAND (?):
```

Respond Y to the DELETE? prompt to delete a data set and N to cancel the delete function. $DISKUT1 continues prompting for each data set on the volume with the specified prefix.

Note: NA means that the end-of-data pointer and flag in the directory member entry have not been set.

$DISKUT1 - Allocate/Delete/List Directory Data *(continued)*

DNGD — Delete All Data-Type Data Sets Not Starting with Text

Use the DNGD command to delete all data-type data sets that do not start with a specific prefix. DNGD operates in the same manner as DNG except you can only delete data-type data sets.

DNGP — Delete All Programs Not Starting with Text

Use the DNGP command to delete all program-type data sets not starting with a specific prefix. DNGP operates in the same manner as DNG except you can only delete program-type data sets.

EN — End the Program

Use the EN command to end the $DISKUT1 utility.

LA — List All Data Sets

Use the LA command to list all data sets (data- and program-type) on a specific volume.

```
COMMAND (?):  LA

USING VOLUME EDX001
 NAME       TYPE       FIRST RECORD  SIZE      EOD/PGMSZ

LNK52      DATA             64        98          98
CONV       DATA            165        11          11
STG2       DATA            409         3          NA
STG4       DATA            426         3          NA
MSGTST5A   PGM             383        26          24

        8864 FREE RECORDS IN LIBRARY
```

$DISKUT1 - Allocate/Delete/List Directory Data *(continued)*

Notes:

1. EOD is the displacement to the next available record for data-type data sets.

2. NA means that the system has not set the end-of-data pointer and flag in the directory member entry.

To cancel a long list, press the attention key, press the enter key, and enter CA.

LACTS — List All Data Sets in CTS/RBA Mode

Use the LACTS command to list all data sets (data- and program-type) on a specific volume.

For the 4962 disk and the 4963 disk subsystem, the system shows the disk locations of the data sets in CTS format. For the 4967 disk subsystem, as well as DDSK-30 and DDSK-60 disks, the system shows the disk locations of the data sets in RBA format.

Example 1: List all data sets in CTS format. Volume EDX001 resides on a 4963 disk subsystem; the system shows the disk locations of the data set in CTS format.

```
COMMAND (?):  LACTS

USING VOLUME EDX001
  NAME       TYPE       ORG(CTS)       END(CTS)

NOMSG        DATA       0080007        0080012
EDXSTART     DATA       0030115        0040106
LNO          DATA       0050006        0050008
    .
    .
    .
FULLR        PGM        0400010        0440008

      1046 FREE RECORDS IN LIBRARY
```

$DISKUT1 - Allocate/Delete/List Directory Data *(continued)*

Example 2: List all data sets in RBA format. Volume EDX001 resides on a 4967 disk subsystem; the system shows the disk locations of the data sets in RBA format.

```
COMMAND (?):  LACTS

USING VOLUME EDX001
  NAME       TYPE      ORG(RBA)      END(RBA)

LNK52        DATA      3331065       3331164
CONV         DATA      3331168       3331178
FULL         PGM       3331182       3331185
   .
   .
   .
CONVP        PGM       3331179       3331181

      8864 FREE RECORDS IN LIBRARY
```

$DISKUT1 - Allocate/Delete/List Directory Data *(continued)*

LAD — List Data Sets on All Volumes

Use the LAD command to list all data sets (data- and program-type) on all volumes. You may only want to list data sets starting with a specific prefix. Do this by entering the LAD command followed by the prefix. This command is useful in finding a data set when you do not know the name of the volume where it resides or if the same data set appears on multiple volumes. $DISKUT1 lists the name of each data set along with the following information:

- the type (data or program)
- the number of the first record in the data set
- the size of the data set
- the last record in a data-type data set or the number of records in a program-type data set
- the volume where it resides.

Example: List data sets with a prefix of 'S' in all volumes.

```
COMMAND (?): LAD S

NAME      TYPE    FIRST RECORD    SIZE     EOD/PGMSZ      VOLUME

SEW       DATA            1646     100            79       EDX002
SSRC      DATA            7748       5             5       EDX002
SMODUL    DATA            1386      10            10       ASMLIB
SWORK     DATA           15522     300           225       EDX003
SDATA     DATA            6989       5             5       EDX005

USING VOLUME EDX005

COMMAND (?):
```

$DISKUT1 - Allocate/Delete/List Directory Data *(continued)*

LAV — List All Volumes

Use the LAV command to list all volumes on your Series/1. The LAV command scans all existing volumes and $DISKUT1 lists the name of each volume along with the following information:

- the device address
- the number of the first record on the volume
- the size of the volume.

When it finishes the scan, $DISKUT1 points to the last volume accessed.

Example:

```
COMMAND (?):  LAV

 VOLUME NAME    DEVICE ADDRESS    FIRST RECORD     SIZE

 EDX002                   0003             361     9480
 ASMLIB                   0003            9841     8000
 MTMSRC                   0013             361     8400
 TSTSRC                   0013            8761     2200

USING VOLUME EDX002

COMMAND (?):
```

To cancel a long list, press the attention key and enter $C $DISKUT1. This cancels the listing and $DISKUT1.

$DISKUT1 - Allocate/Delete/List Directory Data *(continued)*

LD — List Data-Type Data Sets

Use the LD command to list all the data-type data sets on a specific volume, the number of the first record in the data set, and the size of the data set.

Example: List the data sets on volume EDX001.

```
COMMAND (?):  LD

USING VOLUME EDX001

 NAME       FIRST RECORD      SIZE      EOD/PGMSZ
TEXTWORD          105           22             22
$SAMDATA          127           36             30
$NAME3            450           10             NA

         506 FREE RECORDS IN LIBRARY

COMMAND (?):
```

Notes:

1. EOD is the displacement to the next available record.
2. NA means that the system has not set the end-of-data pointer and flag in the directory member entry.

LDCTS — List Data-Type Data Sets in CTS/RBA Mode

Use the LDCTS command to list only data-type data sets on a specific volume. Depending upon the disk on which the volume resides, the system shows the locations of the data sets in CTS or RBA format.

For the 4962 disk and the 4963 disk subsystem, the system shows the disk locations of the members in CTS format. For the 4967 disk subsystem, as well as DDSK-30 and DDSK-60 disks, the system shows the disk locations of the members in RBA format.

$DISKUT1 - Allocate/Delete/List Directory Data *(continued)*

Example 1: List all data sets in CTS format. Volume EDX001 resides on a 4963 disk subsystem; the example shows the disk locations of the data sets in CTS format.

```
COMMAND (?):  LDCTS

USING VOLUME EDX001
  NAME       ORG(CTS)       END(CTS)

NOMSG        0080007        0080012
EDXSTART     0030115        0040106
LNO          0050006        0050008
   .
   .
   .
FULLR        0290105        0290108

      1046 FREE RECORDS IN LIBRARY
```

Example 2: List all data sets in RBA format. Volume EDX001 resides on a 4967 disk subsystem; the example shows the disk locations of the data sets in RBA format.

```
COMMAND (?):  LDCTS

USING VOLUME EDX001
  NAME       ORG(RBA)       END(RBA)

LNK52        3331065        3331164
LNK52U       3331190        3331289
STG1         3331165        3331166
   .
   .
   .
LNK52R       3330663        3330764

      8864 FREE RECORDS IN LIBRARY
```

$DISKUT1 - Allocate/Delete/List Directory Data *(continued)*

LISTP — Direct Listing to $SYSPRTR

Use the LISTP command to direct all $DISKUT1 listings to the device designated as the $SYSPRTR. Once you specify this command, the system directs all listings to $SYSPRTR until you specify another device.

Example: List all subsequent $DISKUT1 listings to the $SYSPRTR.

```
COMMAND (?):  LISTP

COMMAND (?):
```

LISTT — Direct Listing to Terminal

Use the LISTT command to direct all $DISKUT1 listings to the terminal that invoked $DISKUT1. Once you specify this command, the system directs all listings to that terminal until you specify another device.

Example: List all subsequent $DISKUT1 listings to the terminal that invoked $DISKUT1.

```
COMMAND (?):  LISTT

COMMAND (?):  LD EDX002

USING VOLUME EDX002
  NAME   FIRST RECORD  SIZE  EOD/PGMSZ

      2791 FREE RECORDS IN LIBRARY
COMMAND (?):
```

LM — List a Specific Data Set

Use the LM command to list the description of a specific data set (data- or program-type). $DISKUT1 lists the data set type, the disk location of the first record, the size of the data set and the EOD. The EOD is the displacement to the next available record for data-type data sets.

$DISKUT1 - Allocate/Delete/List Directory Data *(continued)*

Example 1: List the directory description of a data-type data set (member).

For the 4962 disk and the 4963 disk subsystem, the system shows the disk location of a data set in CTS format.

```
COMMAND (?):  LM
MEMBER NAME:  TEST12

USING VOLUME EDX001

  NAME     TYPE      FIRST RECORD    SIZE    EOD/PGMSZ

TEST12    DATA            305          7           5

IODA=003 (HEX), CTS=0220115,0220121

COMMAND (?):
```

Notes:

1. IODA=I/O device address; CTS=cylinder, track, and sector. In this example, the data set is on the device at device address 003 at cylinder 22, track 01, from sector 16 through sector 21.

2. FIRST RECORD is the number containing the first record of the data set.

Example 2: List the directory description of a program-type data set (member).

For the 4967 disk subsystems, as well as the DDSK-30 and DDSK-60 disks, the system shows the disk location of a data set in RBA (relative block address) format. In this example, the data set is on the device at address 0048 at RBA 0751100,0760107.

```
COMMAND (?):  LM
MEMBER NAME:  $EDXNUC

USING VOLUME EDX002

  NAME       TYPE      FIRST RECORD    SIZE    EOD/PGMSZ

$EDXNUC     PGM OVLY        121         400          202

IODA=0048 (HEX), RBA=0751100,0760107

COMMAND (?):
```

$DISKUT1 - Allocate/Delete/List Directory Data *(continued)*

Notes:

1. IODA=I/O device address; RBA=relative block address. In this example, the data set is on the device at device address 00C0 at relative block address 0751100.

2. FIRST RECORD is the number containing the first record of the data set.

LP — List Programs

Use the LP command to list all program-type data sets on a specific volume or only those starting with a specified prefix. The system directs the listing to the $SYSPRTR. You can redirect the listing by specifying the label of the printer following the LP command or the prefix. For example, to direct the listing in the example to a printer other than $SYSPRTR, enter the following:

```
COMMAND (?):  LP $DISK $SYSLOGA
```

The system directs the listing, in this case, towards the printer designated as the alternate logging device ($SYSLOGA).

Example: List directory description of the program-type data sets beginning with the prefix $DISK.

```
COMMAND (?):  LP $DISK

USING VOLUME EDX001

  NAME      FIRST RECORD      SIZE      EOD/PGMSZ

$DISKUT1          256           32             25
$DISKUT2          288           30             23
$DISKTST  OVLY    323           15             10

        2550 FREE RECORDS IN LIBRARY

COMMAND (?):
```

Notes:

1. FIRST RECORD is the number of the first record of the data set.

2. PGMSZ shows the size of the program in records, excluding RLDs and overlays.

3. OVLY indicates an overlay program.

$DISKUT1 - Allocate/Delete/List Directory Data *(continued)*

LPCTS — List Program-Type Data Sets in CTS/RBA Mode

Use the LPCTS command to list only program-type data sets on a specific volume. Depending upon the disk on which the volume resides, the location of the data sets (members) are shown in CTS or RBA format.

For the 4962 disk and the 4963 disk subsystem, the system shows the disk locations of the members in CTS format.

For the 4967 disk subsystems, as well as DDSK-30 and DDSK-60 disks, the system shows the disk locations of the members in RBA format.

Example 1: List all data sets in CTS format. Volume EDX001 resides on a 4963 disk subsystem; the system shows the locations of the data sets in CTS format.

```
COMMAND (?):  LPCTS

USING VOLUME EDX001
  NAME       ORG(CTS)       END(CTS)

FULL         0140113        0150001

        1046 FREE RECORDS  IN LIBRARY
```

Example 2: List all data sets in RBA format. Volume EDX001 resides on a 4967 disk subsystem; the system shows the locations of the data sets in RBA format.

```
COMMAND (?):  LPCTS

USING VOLUME EDX001
  NAME       ORG(RBA)       END(RBA)

CONVP       3331179      3331181
FULL        3331182      3331185
   .
   .
   .
FULLR       3340635      3340662
T5A         3340765      3340796

      8864 FREE RECORDS  IN LIBRARY
```

$DISKUT1 - Allocate/Delete/List Directory Data *(continued)*

LS — List Free Space

Use the LS command to list the free space available on a specific volume. In addition, $DISKUT1 lists the following information:

- the size (in records) of the volume
- the number of unused records
- the number of directory entries
- the number of unused directory entries
- the total number of data sets
- the number of free space entries.

$DISKUT1 then prompts if you wish to list the free space chain. Respond Y and $DISKUT1 lists the size and the location (number of the first record) of each area of free space on the volume.

Example: List free space available on volume EDX001.

```
COMMAND (?):  LS

USING VOLUME EDX001

LIBRARY
  AT RECORD          1
  SIZE            3600 RECORDS
  UNUSED           665 RECORDS

DIRECTORY
  SIZE      405 MEMBERS ( 51 RECORDS)
  UNUSED    179 MEMBERS

NUMBER OF MEMBERS -    35

NUMBER OF FREE SPACE ENTRIES -    2

LIST FREE SPACE CHAIN?   Y

 FIRST RECORD  SIZE
        3000    600
        247      65

COMMAND (?):
```

Note: FIRST RECORD is the number of the first record within the data set.

$DISKUT1 - Allocate/Delete/List Directory Data *(continued)*

RE — Rename a Data Set

Use the RE command to rename a data set.

Example: Rename a data set named PROG1 to MYPROG.

```
COMMAND (?):  RE
MEMBER NAME:  PROG1
NEW NAME:  MYPROG
RENAME COMPLETED
```

SE — Set End of Data Pointer/Flag

Use the SE command to set the end-of-data pointer/flag on within a data- or program-type data set.

Example: Set end of data pointer for data set named DATAFILE.

```
COMMAND (?):  SE DATAFILE

DATAFILE EOD POINTER IS NOW       :  0
DATAFILE EOD FLAG IS NOW OFF

NEW EOD (-1 TO RESET EOD AND FLAG):  2
ARE ALL PARAMETERS CORRECT?  Y

DATAFILE EOD POINTER IS NOW       :  2
DATAFILE EOD FLAG IS NOW ON

COMMAND (?): SE DATAFILE

DATAFILE EOD POINTER IS NOW       :  2
DATAFILE EOD FLAG IS NOW          :  ON

NEW EOD (-1 TO RESET EOD AND FLAG):  -1
ARE ALL PARAMETERS CORRECT ?  Y

DATAFILE EOD POINTER IS NOW       :  0
DATAFILE EOD FLAG IS NOW          :  OFF

COMMAND (?):
```

Notes:

1. RESET option sets the EOD flag off and the EOD pointer to 0.
2. This command modifies fields within the directory member entry. It does not change the actual data set.

$DISKUT1 - Allocate/Delete/List Directory Data *(continued)*

SNQ — Reset Prompt Modes

Use the SNQ command to turn off the prompt mode for the delete generic commands (DG, DGP, DGD, DNG, DNGP and DNGD). This means that for these commands, $DISKUT1 deletes the data sets without prompting you to verify each one. If you want to turn prompt mode off, do so before you use any of these commands.

```
COMMAND (?): SNQ
```

SQ — Set Prompt Modes

Use the SQ command to set $DISKUT1 to prompt mode for the delete generic commands. This means that for the DG, DPG, DGD, DNG, DNGP and DNGD commands, $DISKUT1 prompts you to verify each data set you want to delete. Prompt mode is the default for the list commands. If you do not want to be prompted for each data set to be deleted, use the SNQ command to turn off prompt mode.

Example:

```
COMMAND (?): SQ
```

$DISKUT2 - Patch/Dump/List Data Set or Program

With $DISKUT2, you can perform the following operations on data sets and/or programs:

- Clear (set to zero) all or portions of a data set and reset the end-of-data pointer.
- Dump any data set created using $EDIT1N or $FSEDIT or any program on the terminal you are using or on the printer of your choice.
- Patch a data record in a data set or an address within a program.
- Modify the default load time storage allocation associated with a program.
- List the contents of a data set on the terminal you are using or on the printer of your choice.
- List the log data set associated with a specific device on the terminal you are using or on the printer of your choice.

  Note: For tape management functions, see "$TAPEUT1 - Tape Management" on page UT-522.

Program and Data Set Member Dumps and Patches

You make program member dumps and patches by relative address (hexadecimal) within the program. The relative address corresponds exactly to the address specified in the LOC field of an assembly listing. You can enter data in decimal, hexadecimal, or EBCDIC as shown in the examples that follow.

You make data set member dumps and patches by specifying a record number and a first word. The numbering for both record and word number begins with 1. You can enter data in either decimal, hexadecimal, or EBCDIC. You should separate each field of patch data with a nonnumeric character other than a carriage return.

Note: Any patch you make to a data set or a program is permanent. Be sure that the data set record or program address you are patching is correct. Check a dump of the data set or the program assembly listing before you perform a patch.

The system formats dumps of program or data set members when you select hexadecimal as an option.

$DISKUT2 - Patch/Dump/List Data Set or Program *(continued)*

Absolute Record Numbers

A special feature of $DISKUT2 allows dumping and/or patching of any area on a disk volume by referencing absolute record numbers. Select this this mode by entering the characters $$EDXVOL as a member data set name. When you use this mode, the system will direct operation to absolute record numbers rather than symbolic data/program member names, with record 1 being the first physical record on the disk or diskette where the volume resides.

If you enter the special system name $$EDXLIB, the system uses absolute record numbers and considers the first record in the directory as record 1. $$EDXVOL references the first physical record on the disk or diskette. On diskettes, the system uses $$EDXVOL to reference records on cylinder 0 only (if you attempt to access other cylinders, you will produce unpredictable results). You can reference all other records on diskette using $$EDXLIB.

Notes:

1. $$, $$EDXVOL, and $$EDXLIB are special system data set names. $$ is a reserved system name. $$EDXVOL points to the beginning of a volume. $$EDXLIB points to the beginning of the data set directory within a volume.

2. When using this mode, you also have access to records that are meant for Series/1 hardware use only. For example, the system designates records 121 through 240 on the 4962 disk for alternate sector assignment and they are not meant to be accessed directly.

3. When you use the DU or LU commands to dump or list $$EDXVOL, you dump only the data, not the whole device.

Invoking $DISKUT2

You invoke $DISKUT2 with the $L command or option 3.2 of the session manager.

```
> $L $DISKUT2
LOADING $DISKUT2    51P,00:30:15, LP= 9200, PART=1

$DISKUT2 - DATA SET MGMT. UTILITY II
USING VOLUME XXXXXXX
COMMAND (?):
```

$DISKUT2 - Patch/Dump/List Data Set or Program *(continued)*

$DISKUT2 Commands

To display the $DISKUT2 commands at your terminal, enter a question mark in response to the prompting command COMMAND (?):.

```
COMMAND (?):  ?

CD - CLEAR DATA SET
CV - CHANGE VOLUME
DP - DUMP DS OR PGM ON PRINTER... DP DSNAME PRTNAME
DU - DUMP DS OR PGM ON CONSOLE
   (-CA- WILL CANCEL)
PA - PATCH DS OR PGM
SS - SET PROGRAM STORAGE PARM
LP - LIST DS ON ANY PRINTER... LP DSNAME PRTNAME
LU - LIST DS ON CONSOLE
PL - LIST LOG ON ANY PRINTER... PL DSNAME PRTNAME
LL - LIST LOG ON CONSOLE
PR - LIST LOG BY WRAP COUNT AND RELATIVE RECORD ON ANY PRINTER
LR - LIST LOG BY WRAP COUNT AND RELATIVE RECORD ON CONSOLE
EN - END PROGRAM

COMMAND (?):
```

Note: The LR and PR commands are for remote manager (RM1) users only.

$DISKUT2 includes the time and the date in your listing if you include timer support in your system and you direct output to a print device using DP, LP, PL,, or PR. You can send your output to any printer using the PRTNAME parameter on the DP, LP, PL, or PR command. The default is $SYSPRTR.

Note: If you dump (DU) or list (LU) a data set on a terminal using $$EDXVOL, you are limited to the number of logical records.

All the functions listed (except for ?, CV, and EN) act upon the IPL volume. When you invoke $DISKUT2, it issues the following message:

```
USING VOLUME volname
```

To point to another volume to perform one or more of the previously listed functions, enter the CV command and the name of the volume.

```
COMMAND (?):  CV volname
USING VOLUME volname
```

$DISKUT2 - Patch/Dump/List Data Set or Program *(continued)*

All functions act upon the specified volume until you change them with another CV command or until you end and reload $DISKUT2. If you specify an invalid volume a CV command, the utility uses the IPL volume, if it is available. If the IPL volume is not available, the system issues the message NO VOLUME AVAILABLE. You can either end the $DISKUT2 or change volumes using the CV command.

Each command and its explanation is presented in alphabetical order on the following pages.

CD — Clear a Data Set (to Zeros)

Use CD to clear an entire data set or a portion of a data set and to reset the end-of-data pointer. This sets the data within the data set to zero.

Example:

```
COMMAND (?): CD
DATA SET NAME?  DATA
CLEAR ENTIRE DATA SET? N
FIRST   RECORD: 1
LAST    RECORD: 100

RESET THE E.O.D. POINTER? Y
HOW MANY RECORDS TO EOD?  100

ARE ALL PARAMETERS CORRECT? Y
CLEAR COMPLETED

COMMAND (?):
```

CV — Change Volume

Use the CV command to change volumes. When you invoke $DISKUT2, it assumes you are using the IPL volume. All $DISKUT2 functions operate on the IPL volume until you change to another volume.

Example:

```
USING VOLUME EDX002

COMMAND (?): CV EDX40
USING VOLUME EDX40

COMMAND (?):
```

$DISKUT2 - Patch/Dump/List Data Set or Program *(continued)*

DP — Dump a Data Set or Program on a Printer

Use DP to dump all or portions of a data set or program to a printer. If you don't specify a printer, the system directs the dump to the $SYSPRTR.

Example 1: Dump a data set on a printer other than $SYSPRTR.

```
COMMAND (?):  DP EJW1 MPRTR
EJW1 IS A DATA SET
FIRST RECORD (0 TO CANCEL COMMAND):  1
LAST RECORD:  2
FIRST WORD:  1
WORDS/RECORDS:  12
(D)EC, (E)BCDIC OR (H)EX:  H

DUMP COMPLETE
ANOTHER AREA?  N

COMMAND (?):
```

Notes:

1. You must specify the printer name on the same line as the command and the data set/program. The system does not issue a prompt for the printer name.

2. The printer name is the label on the TERMINAL definition statement defining the printer to the supervisor.

Example 2: Dump a portion of a program on $SYSPRTR.

```
COMMAND (?):  DP
PGM OR DS NAME:  MYPROG
MYPROG IS A PROGRAM OF HEX SIZE 0000026C
ADDRESS:  0000
DUMP TO PROGRAM END?  N
HOW MANY WORDS?  20

 0000     0008 0709 D6C7 D9C1 D440 0000 01B4 0254    |..PROGRAM ...M..|
 0010     0000 0000 0258 0000 0000 0000 0100 0256    |................|
 0020     0000 0000 0000 0000                        |.........       |

DUMP COMPLETE
ANOTHER AREA?  N
COMMAND (?):
```

$DISKUT2 - Patch/Dump/List Data Set or Program *(continued)*

DU — Dump a Data Set or Program on Terminal

Use the DU command to dump all or a portion of a data set or a program to the terminal where you invoked $DISKUT2.

Example 1: Dump a portion of a data set on the terminal.

```
COMMAND(?): DU
PGM OR DS NAME: EDITWORK
EDITWORK IS A DATA SET
FIRST RECORD (0 TO CANCEL COMMAND): 1
LAST RECORD: 1
FIRST WORD: 1
WORDS / RECORD: 52
(D)EC, (E)BCDIC, OR (H)EX: H

RECORD 1
  1  7A2E 78D0 0088 7A30 000A 101A D240 4040  |:......:..
  9  0000 0000 34D6 0000 0000 0000 0000 FFFF  |.....O...
 17  0000 0000 14D0 5600 0000 7A02 0000 0000  |.........
 25  0000 0000 0000 0000 0000 0000 0000 0000  |.........
 33  0000 0000 0000 0000 5040 6F03 023C 0254  |........&
 41  7B96 402F 7BA0 0000 182C 6808 00C4 680D  |#. .#....
 49  7BA4 6808 00F6 680D                      |#....6..

DUMP COMPLETE
ANOTHER AREA?
COMMAND (?):
```

Example 2: Dump a portion of a program on the terminal.

```
COMMAND (?): DU
PGM OR DS NAME: EX31DS02
EX31DS02 IS A PROGRAM OF HEX SIZE 00003B7E
ADDRESS:  100
DUMP TO PROGRAM END? N
HOW MANY WORDS? 20

 0100    C4C1 E3C1 F0F2 4040 0606 C4C9 E2D2 F0F2 |DATA02  ..DISK02|
 0110    0000 0000 0000 0000 0001 0000 0001 0000 |................|
 0120    0000 0000 0000 0000                     |........        |

DUMP COMPLETE
ANOTHER AREA? N

COMMAND (?):
```

$DISKUT2 - Patch/Dump/List Data Set or Program *(continued)*

Example 3: Dump a portion of a program with overlay segments on the terminal.

```
COMMAND (?): DU
PGM OR DS NAME: EX31ES01
EX31ES01 IS A PROGRAM OF HEX SIZE 00002FA8 WITH 2 OVERLAY SEGMENTS
ADDRESS: 100
HOW MANY WORDS? 10

 0100   0000 00B2 C026 2221 7CC4 D640 E8D6 E440 |........@DO YOU |
 0110   E6C1 D5E3                               |WANT            |

DUMP COMPLETE
ANOTHER AREA? N

COMMAND (?):
```

Example 4: Dump an overlay segment of a program on a terminal.

```
COMMAND (?): DU EX31ES01
EX31ES01 IS A PROGRAM OF HEX SIZE 00002FA8 WITH 2 OVERLAY SEGMENTS
ADDRESS: 2F00
DUMP OVERLAY SEGEMENT (O) OR RESIDENT CODE (R)? O
WHICH OVERLAY? 2
HOW MANY WORDS? 10

 0258   3232 C9D5 C4C5 D7C5 D5C4 C5D5 E340 D7D9 |..INDEPENDENT PR|
 0268   D6C7 D9C1                               |OGRA            |

DUMP COMPLETE
ANOTHER AREA? N

COMMAND (?):
```

Note: Addresses within an overlay segment are relative to the beginning of that overlay segment.

$DISKUT2 - Patch/Dump/List Data Set or Program *(continued)*

Example 5: Dump a portion of the supervisor that is located in an overlay area.

```
COMMAND (?): DU
PGM OR DS NAME: $EDXNUCC

ENTER PARTITION NUMBER(1-8) : 1
PARTITION 1 OF $EDXNUCC HAS A HEX SIZE OF 0000D9FF WITH 3 OVERLAY SEGMENTS
ADDRESS: D000
DUMP OVERLAY SEGMENT (O) OR RESIDENT CODE (R)? O
WHICH OVERLAY? 2
HOW MANY WORDS?3

0304    10A2 0420 041E                        |..

DUMP COMPLETE
ANOTHER AREA? N

COMMAND (?):
```

Example 6: Dump a portion of a supervisor with overlay segments. The portion you are dumping does not reside within the overlay segment.

```
COMMAND (?): DU
PGM OR DS NAME: $EDXNUCC

ENTER PARTITION NUMBER(1-8) : 1
PARTITION 1 OF $EDXNUCC HAS A HEX SIZE OF 0000D9FF WITH 3 OVERLAY SEGMENTS
ADDRESS: D000
DUMP OVERLAY SEGMENT (O) OR RESIDENT CODE (R) ? R
HOW MANY WORDS? 3

D000    0000 0000 0000                        |..

DUMP COMPLETE
ANOTHER AREA? N

COMMAND (?):
```

$DISKUT2 - Patch/Dump/List Data Set or Program *(continued)*

Example 7: Dump a portion of a supervisor not located within an overlay area.

```
COMMAND (?):  DU
PGM OR DS NAME:  $EDXNUCC

ENTER PARTITION NUMBER(1-8):  1
PARTITION 1 OF $EDXNUCC HAS A HEX SIZE OF 0000D9FF WITH 3 OVERLAY SEGMENTS
ADDRESS:  9000
HOW MANY WORDS?  3

9000    0000 0004 8C60                      |..

DUMP COMPLETE
ANOTHER AREA?  N

COMMAND (?):
```

You can use single-line entry; however, be sure to enter the information required in the order that the $DISKUT2 expects it. Here is the information for the above example entered in single-line entry:

```
COMMAND (?):  DU $EDXNUCC 1 9000 3
```

EN — End $DISKUT2

Use EN to end $DISKUT2.

Example

```
COMMAND (?):  EN
$DISKUT2 ENDED AT 00:36:04
```

$DISKUT2 - Patch/Dump/List Data Set or Program *(continued)*

LL — List Log Data Set for a Specific Device on a Terminal

Use LL to list the log data set for a specific device on the terminal where you invoked $DISKUT2.

Example: List log data set for device at address 002 on the terminal. (Refer to the *Problem Determination Guide* for an explanation of the log output.)

```
COMMAND (?): LL
LOG DS NAME: $LOGDS
ENTER DEVICE ADDRESS, NULL FOR ALL ENTRIES,
 OR 'FFFF' FOR PROGRAM/SYSTEM CHECKS ONLY: 002
ERROR LOG LIST, DATA SET: LOGDS ON EDX002
ADDR: 0002
I/O LOG ERROR COUNTERS (BY DEVICE ADDR):

0000    0000 0200 0000 0000 0000 0000 0000 0000
0010    0000 0000 0000 0000 0000 0000 0000 0000
0020    0000 0000 0004 0000 0000 0000 0000 0000
....    .... .... .... .... .... .... .... ....
....    .... .... .... .... .... .... .... ....
00E0    0000 0000 0000 0000 0000 0000 0000 0000
00F0    0000 0000 0000 0000 0000 0000 0000 0000

SOFT RECOV. ERR
DEV ADDR: 0002      DEV ID: 0106
DATE: 7/ 7/80       LVL: 0001   AKR: 0000
TIME: 11:13:20      RETRY: 3    IDCB: 7002 0874
INTCC: 0012         ISB: 0080
DCB1: 8007 0000 0000 0000 0000 0884 0000 0000
DCB2: 8005 0001 0000 0001 0000 0894 0000 0000
DCB3: 2009 0000 0000 0001 0001 0000 0100 CBEE
CSSW: 08A3 0400 0001 0001
```

$DISKUT2 - Patch/Dump/List Data Set or Program *(continued)*

LP — List All or a Portion of a Data Set on Printer

Use LP to list all or a portion of a data set on the printer. If you do not specify a printer, the system directs the list to the $SYSPRTR.

Example 1: List a data set on a printer other than $SYSPRTR.

```
COMMAND (?):  LP CS MPRTR
LIST ALL OF THE DATA SET?  Y

COMMAND (?):
```

Notes:

1. You must specify the printer name on the same line as the command and the data set/program. The system does not issue a prompt for the printer name.

2. The printer name is the label on the TERMINAL definition statement defining the printer to the supervisor.

Example 2: List a portion of a data set on $SYSPRTR.

```
COMMAND (?):  LP
DATA SET NAME?  MYPROG
LIST ALL OF THE DATA SET?  N
FIRST RECORD:  1
LAST RECORD:  20

LIST COMPLETE

COMMAND (?):
```

$DISKUT2 - Patch/Dump/List Data Set or Program *(continued)*

LR — List Log by Wrap Count and Relative Record for a Specific Device on a Terminal

Use LR to list the log by wrap count and relative record on the terminal where you invoked $DISKUT2.

Example 1: List log by wrap count and relative record on current terminal. (Refer to the *Problem Determination Guide* for an explanation of the log output.)

```
COMMAND (?): LR
LOG DATA SET (NAME,VOLUME): $LOGDS,EDX002

ENTER (HEX) WRAP COUNT AND RELATIVE
RECORD NUMBER IN THE FORM WWWW RRRR: 0001 0001
                   .
                   .
                   .
SOFT RECOV. ERR
DEV ADDR:  0002         DEV ID:  0106
DATE:  12/10/83         LVL:  0001   AKR:  0000
TIME:  11:10:20         RETRY:  3    IDCB:  7002 0874
INTCC:  0012            ISB:  0080
DCB1:  8007 0000 0000 0000 0000 0884 0000 0000
DCB2:  8005 0001 0000 0001 0000 0894 0000 0000
DCB3:  2009 0000 0000 0001 0001 0000 0100 CBEE
CSSW:  08A3 0400 0001 0001
```

When your log data set is exactly filled, $LOG increments its wrap count by one even if there is no data after the wrap. The header, therefore, shows a wrap with no data. In this situation, specify the previous wrap count number. By doing so, you dump the entire data set.

$DISKUT2 - Patch/Dump/List Data Set or Program *(continued)*

Example 2: In the following example of a log eight records long, two are control records and six are data records. $LOG writes to the end of the data set and sets the wrap count to two. No more data is written.

```
COMMAND (?):  LR
LOG DATA SET (NAME,VOLUME):  $LOGDS,EDX002

ENTER (HEX) WRAP COUNT AND RELATIVE
RECORD NUMBER IN THE FORM WWWW RRRR:  2

*** LOG DATA SET WRAP COUNT IS 0002 HEX
*** NO DATA WAS WRITTEN AFTER THE LAST WRAP
*** ENTER PREVIOUS WRAP COUNT - 0001 HEX

ENTER (HEX) WRAP COUNT AND RELATIVE
RECORD NUMBER IN THE FORM WWWW RRRR:  1

*** RELATIVE RECORD OUT OF RANGE.
*** ALLOWABLE RECORDS ARE 0001 - 0006 HEX

ENTER (HEX) WRAP COUNT AND RELATIVE
RECORD NUMBER IN THE FORM WWWW RRRR:  1 6
```

LU — List Contents of a Data Set on Terminal

Use LU to list all or a portion of a source data set on the terminal where you invoked $DISKUT2.

Example: List a portion of a data set.

```
COMMAND(?):  LU
DATA SET NAME?  CALSRC
LIST ALL OF THE DATA SET?  N
FIRST RECORD:  4
LAST RECORD:  8

ATTNLIST ATTNLIST  (STOP,POST1,CALC,POST2)
         SPACE 1
POST1    POST      KBEVENT,1
         ENDATTN
         SPACE 1

LIST COMPLETE

COMMAND (?):
```

$DISKUT2 - Patch/Dump/List Data Set or Program *(continued)*

PA — Patch a Data Set or Program

Use PA to patch a data set record(s) or an area within a program in decimal, EBCDIC, or hexadecimal.

Example 1: Patch a data set in decimal.

```
COMMAND (?): PA ASMOBJ
ASMOBJ IS A DATA SET
FIRST RECORD (0 TO CANCEL COMMAND): 2
FIRST WORD: 3
HOW MANY WORDS? 4
(D)EC, (E)BCDIC OR (H)EX? D

NOW IS:
      3      16

ENTER DATA: 18

NEW DATA:
      3      18

OK? Y
PATCH COMPLETE
ANOTHER PATCH? N

COMMAND (?):
```

Example 2: Patch a data set in EBCDIC.

```
COMMAND (?): PA ASMOBJ
ASMOBJ IS A DATA SET
FIRST RECORD (0 TO CANCEL COMMAND): 2
FIRST WORD: 3
HOW MANY WORDS? 4
(D)EC, (E)BCDIC OR (H)EX? E

NOW IS:
      3      0012

ENTER DATA: 0010

NEW DATA:
      3      F0F0

OK? Y
PATCH COMPLETE
ANOTHER PATCH? N

COMMAND (?):
```

$DISKUT2 - Patch/Dump/List Data Set or Program *(continued)*

Example 3: Patch a data set in hexadecimal.

```
COMMAND (?):  PA ASMOBJ
ASMOBJ IS A DATA SET
FIRST RECORD (0 TO CANCEL COMMAND):  2
FIRST WORD:  3
HOW MANY WORDS?  4
(D)EC, (E)BCDIC OR (H)EX?  H

NOW IS:
     3      0000 0000 0000 0030          |.........     |

ENTER DATA:  0002 0003 0004 000A

NEW DATA:
     3      0002 0003 0004 000A          |.........      |

OK?  Y
PATCH COMPLETE
ANOTHER PATCH?  N

COMMAND (?):
```

Example 4: Patch a program in decimal.

```
COMMAND (?):  PA
PGM OR DS NAME:  MYPROG
MYPROG IS A PROGRAM OF HEX SIZE 000013FE
ADDRESS:  02D8
HOW MANY WORDS?  1
(D)EC, (E)BCDIC, OR (H)EX?:  D

NOW IS:
 02D8  8

ENTER DATA:  6

NEW DATA:
 02D8  6

OK?  Y
PATCH COMPLETE
ANOTHER PATCH?  N

COMMAND (?):
```

$DISKUT2 - Patch/Dump/List Data Set or Program *(continued)*

Example 5: Patch a program in EBCDIC.

```
COMMAND (?): PA
PGM OR DS NAME: MYPROG
MYPROG IS A PROGRAM OF HEX SIZE 000013FE
ADDRESS: 2D8
HOW MANY WORDS? 7
(D)EC, (E)BCDIC, OR (H)EX?: E

NOW IS:
 02D8  D4C5 D4C2 C5D9 40C4 C5D3 C5.. ..  | MEMBER DELETE@

ENTER DATA: DELETE MEMBER@

NEW DATA:
 02D8  C4C5 D3C5 E3C5 40D4 C5D4 C2.. ..  | DELETE MEMBER@

OK? Y
PATCH COMPLETE
ANOTHER PATCH? N

COMMAND (?):
```

Example 6: Patch a program in hexadecimal.

```
COMMAND (?): PA EX31DS02
EX31DS02 IS A PROGRAM OF HEX SIZE 00003B7E
ADDRESS: 37B
HOW MANY WORDS? 3
(D)EC, (E)BCDIC, OR (H)EX?: H

NOW IS:
 037A      0000 0001 005D              |.....)       |

ENTER DATA: 0020 0003 005F

NEW DATA:
 037A      0020 0003 005F              |.....¬        |

OK? Y
PATCH COMPLETE
ANOTHER PATCH? N

COMMAND (?):
```

$DISKUT2 - Patch/Dump/List Data Set or Program *(continued)*

Example 7: Patch a program with overlay segments.

```
COMMAND (?):    PA EX31ES01
EX31ES01 IS A PROGRAM OF HEX SIZE 00002FA8 WITH 2 OVERLAY SEGMENTS
ADDRESS:    2F32
PATCH OVERLAY SEGMENT (O) OR RESIDENT CODE (R)?    O
WHICH OVERLAY?    1
HOW MANY WORDS?    2
(D)EC, (E)BCDIC, OR (H)EX?:    H

NOW IS:
 028A      0001 009A                  |....          |

ENTER DATA:    0002 009B

NEW DATA:
 028A      0002 009B                  |....          |

OK?    Y
PATCH COMPLETE
ANOTHER PATCH?    N

COMMAND (?):
```

Note: Addresses within an overlay segment are relative to the beginning of that overlay segment.

Example 8: Patch a portion of the supervisor that resides in an overlay area.

```
COMMAND (?):    PA
PGM OR DS NAME:    $EDXNUCQ

ENTER PARTITION NUMBER(1-8):    1
PARTITION 1 OF $EDXNUCC HAS A HEX SIZE OF 0000D9FF WITH 3 OVERLAY SEGMENTS
ADDRESS:  D000
PATCH OVERLAY SEGMENT (O) OR RESIDENT CODE (R)?    O
WHICH OVERLAY?    2
HOW MANY WORDS?    3
(D)EC, (E)BCDIC, OR (H)EX?:    X

NOW IS:
 0304      10A2 0420 041E             |....          |

ENTER DATA:

NEW DATA:
 0304      0000 0000 0000             |....          |

OK?    N
ANOTHER PATCH?    N

COMMAND (?):
```

$DISKUT2 - Patch/Dump/List Data Set or Program *(continued)*

Example 9: Patch a portion of the supervisor that resides in partition 2.

```
COMMAND (?): PA
PGM OR DS NAME: $EDXNUCQ

ENTER PARTITION NUMBER(1-8): 2
PARTITION 2 OF $EDXNUCQ HAS A HEX SIZE OF   000028FE
ADDRESS: 30
HOW MANY WORDS? 2
(D)EC, (E)BCDIC, OR (H)EX?: X

NOW IS:
 0030      40EF 0020                    |....           |

ENTER DATA:

NEW DATA:
 0030      0000 0000                    |....           |

OK? N
```

You can use single-line entry; however, be sure to enter the information required in the order that the $DISKUT2 expects it. Here is the information for the above example entered in single-line entry:

```
ANOTHER PATCH? Y 2 30 2 X

NOW IS:
 0030      40EF 0020                    |....           |

ENTER DATA:

NEW DATA:
 0030      0000 0000                    |....           |

OK? Y
ANOTHER PATCH? N

COMMAND (?):
```

$DISKUT2 - Patch/Dump/List Data Set or Program *(continued)*

PL — List Log Data Set for a Specific Device on a Printer

Use PL to list the log data set for a specific device or all devices on the printer of your choice. If you do not specify a printer, the system directs the list to the $SYSPRTR.

Example 1: List the log data set for device 002 on the $SYSPRTR. (Refer to the *Problem Determination Guide* for an explanation of the log output.)

```
COMMAND (?):     PL
LOG DS NAME:     $LOGDS
ENTER DEVICE ADDRESS, NULL FOR ALL ENTRIES
 OR 'FFFF' FOR PROGRAM/SYSTEM CHECKS ONLY:     002
ERROR LOG LIST, DATA SET:    LOGDS ON EDX002
ADDR:    0002
I/O LOG ERROR COUNTERS (BY DEVICE ADDR):

0000     0000 0200 0000 0000 0000 0000 0000 0000
0010     0000 0000 0000 0000 0000 0000 0000 0000
0020     0000 0000 0004 0000 0000 0000 0000 0000
....     .... .... .... .... .... .... .... ....
....     .... .... .... .... .... .... .... ....
00E0     0000 0000 0000 0000 0000 0000 0000 0000
00F0     0000 0000 0000 0000 0000 0000 0000 0000

SOFT RECOV. ERR
DEV ADDR: 0002      DEV ID: 0106
DATE:   7/ 7/80      LVL: 0001   AKR: 0000
TIME: 11:13:20      RETRY:  3   IDCB: 7002 0874
INTCC: 0012         ISB: 0080
DCB1:  8007 0000 0000 0000 0000 0884 0000 0000
DCB2:  8005 0001 0000 0001 0000 0894 0000 0000
DCB3:  2009 0000 0000 0001 0001 0000 0100 CBEE
CSSW:  08A3 0400 0001 0001
```

$DISKUT2 - Patch/Dump/List Data Set or Program *(continued)*

Example 2: List the log data set for device 002 on a printer other than $SYSPRTR.

```
COMMAND (?): PL $LOGDS MPRTR
DEVICE ADDRE2S(NULL FOR ALL): 002

ERROR LOG LIST, DATA SET: LOGDS ON EDX002
ADDR: 0002
I/O LOG ERROR COUNTERS (BY DEVICE ADDR):

0000    0000 0200 0000 0000 0000 0000 0000 0000
0010    0000 0000 0000 0000 0000 0000 0000 0000
0020    0000 0000 0004 0000 0000 0000 0000 0000
....    .... .... .... .... .... .... .... ....
....    .... .... .... .... .... .... .... ....
00E0    0000 0000 0000 0000 0000 0000 0000 0000
00F0    0000 0000 0000 0000 0000 0000 0000 0000

SOFT RECOV. ERR
DEV ADDR: 0002      DEV ID: 0106
DATE: 7/ 7/80      LVL: 0001   AKR: 0000
TIME: 11:13:20      RETRY: 3   IDCB: 7002 0874
INTCC: 0012         ISB: 0080
DCB1: 8007 0000 0000 0000 0000 0884 0000 0000
DCB2: 8005 0001 0000 0001 0000 0894 0000 0000
DCB3: 2009 0000 0000 0001 0001 0000 0100 CBEE
CSSW: 08A3 0400 0001 0001
```

Notes:

1. You must specify the printer name on the same line as the command and the program name. The system does not issue a prompt for the printer name.

2. The printer name is the label on the TERMINAL definition statement defining the printer to the supervisor.

$DISKUT2 - Patch/Dump/List Data Set or Program *(continued)*

PR — List Log by Wrap Count and Relative Record for a Specific Device on a Printer

If you are a remote manager user (RM1), use PR to list the log wrap count and relative record for a specific device or all devices on the printer of your choice. If you do not specify a printer, the system directs the list to $SYSPRTR.

Example: List log by wrap count and relative record on $SYSPRTR. (Refer to the *Problem Determination Guide* for an explanation of the log output.)

```
COMMAND (?):    PR
LOG DATA SET (NAME,VOLUME):    $LOGDS,EDX002

ENTER (HEX) WRAP COUNT AND RELATIVE
RECORD NUMBER IN THE FORM WWWW RRRR:    0001 0001

ERROR LOG LIST, DATA SET:  $LOGDS  ON EDX002 FOR WRAP COUNT:  0001
                                             AND RECORD # 0001
                              .
                              .
                              .
SOFT RECOV. ERR
DEV ADDR:  0002        DEV ID:  0106
DATE:  12/10/83        LVL:  0001   AKR:  0000
TIME:  11:10:20        RETRY:  3    IDCB:  7002 0874
INTCC:  0012           ISB:  0080
DCB1:  8007 0000 0000 0000 0000 0884 0000 0000
DCB2:  8005 0001 0000 0001 0000 0894 0000 0000
DCB3:  2009 0000 0000 0001 0001 0000 0100 CBEE
CSSW:  08A3 0400 0001 0001
```

If you enter an incorrect wrap count for PR or LR, the system displays the following message:

```
*** INCORRECT WRAP COUNT SPECIFIED.
*** LOG DATA SET WRAP COUNT IS 0001 HEX.
```

If you enter an invalid record number, the system displays the following message:

```
*** RELATIVE RECORD OUT OF RANGE.
*** ALLOWABLE RECORDS ARE 0001 - 0007 HEX.
```

$DISKUT2 - Patch/Dump/List Data Set or Program *(continued)*

SS — Set Program Storage Parameter

Use SS to modify the default load time storage allocation associated with a program. You can change the allocation without reassembling the source code or providing an override on the LOAD instruction. SS requires that you express the size in bytes in decimal. The system rounds up the value you request if it is not an even multiple of 256.

Example: Reduce the dynamic storage you want allocated for the COBOL compiler at program load.

```
> $L $DISKUT2
 LOADING $DISKUT2    51P,01:30:15, LP= 9200, PART=1

 $DISKUT2 - DATA SET MGMT, UTILITY II

 USING VOLUME EDX002
 COMMAND (?): CV ASMLIB
 USING VOLUME ASMLIB

 COMMAND (?): SS $COBOL
 ENTER NEW STORAGE SIZE IN BYTES: 2816

 OLD STORAGE SIZE WAS 8448

 NEW SIZE WILL BE 2816

 OK TO CONTINUE? Y

 COMMAND (?): EN
 $DISKUT2 ENDED AT 01:32:02
```

$DIUTIL - Maintain Partitioned Data Base

$DIUTIL maintains a disk-resident partitioned data base. This utility provides comprehensive facilities to keep the data base current by means of the following functions:

- Initialize the Disk-Resident data base.
- Delete a member.
- Reclaim space in the data base due to deleted members.
- Display contents of data base.
- Copy the data base.
- Copy individual members of the data base.
- Allocate and build a data member.

Normally, you use $DIUTIL only when no other programs of the display processor are in use. You can change the online data base or you may select another data base for the system to reference. This allows you to create displays in a data base other than the online data base and then copy the members into the online data base after testing.

Invoking $DIUTIL

You invoke $DIUTIL with the $L command or option 5.1 of the session manager. To start execution of $DIUTIL:

1. Load $DIUTIL specifying the appropriate data set. You can use $DIFILE, the online data set, or any other data set. However, be sure that another user or program is not changing or using the same data set.

```
 > $L $DIUTIL
DISPLIB (NAME,VOLUME):
```

2. The system responds with the program-loaded message followed by:

```
DISPLAY DATA BASE UTILITY
COMMAND (?):
```

$DIUTIL - Maintain Partitioned Data Base *(continued)*

$DIUTIL Commands

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

```
COMMAND (?):  ?

AL - ALLOCATE DATA MEMBER
BU - BUILD DATA MEMBER
CP - COMPRESS DATA BASE
CM - COPY MEMBER
DE - DELETE A MEMBER
EN - EXIT PROGRAM
IN - INITIALIZE DATA BASE
LA - DISPLAY MEMBER DIRECTORY
LH - DISPLAY MEMBER HEADER
MD - MOVE DATA BASE
RE - RENAME MEMBER
ST - DISPLAY DATA SET STATUS

COMMAND (?):
```

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

AL — Allocate Data Member

Use the AL command to reserve space in a data base for one of several types of data members. The system requests information such as size in sectors and member codes. Member codes are specified as follows:

4 -- Print Report Data Member: The system requests information such as number of lines and line length. It then enters each line, limited to 132 characters each.

5 -- Plot Curve Data Member: The system requests information such as X and Y ranges, X and Y base values, and number of points it must plot. You can select automatic entry of the X points to reduce the data entry requirements. The system provides a sawtooth pattern option to shade under the curve for more vivid presentation of plotted data. Using fewer than 200 points on the X axis gives an inadequate shading effect.

$DIUTIL - Maintain Partitioned Data Base *(continued)*

6 -- Realtime Data Member: The system requests the number of records. You can enter hexadecimal data for testing.

7 thru 9 -- User Data Member: The build function uses these codes to guide you through the correct data entry procedure.

```
COMMAND (?):   AL
MEMBER NAME:   TDATA
ENTER # OF RECORDS TO ALLOCATE?   10
ENTER MEMBER CODE #:   4
MEMBER TDATA ALLOCATED

COMMAND (?):
```

BU — Build Data Member

Use the BU command to insert fixed data into a data member. This command allows you to enter data records to describe a fixed display or enter records, which normally will be dynamic, with a fixed value, to allow testing of the display.

You may have allocated the member using AL; if not, the system prompts you for the allocation information it requires before it proceeds with the :"build:" process. The system guides you one step at a time through the initialization of the data member.

```
COMMAND (?):   BU
ENTER MEMBER NAME:  RDATA
INITIALIZE REPORT DATA MEMBER
ENTER # OF LINES IN REPORT:   2
LINE LENGTH=32
ENTER LINE ITEMS
LINE ONE OF REPORT
LINE TWO OF REPORT
MEMBER LOADED

COMMAND (?):
```

In this case, the member was already allocated.

$DIUTIL - Maintain Partitioned Data Base *(continued)*

CP — Compress Data Base

Use the CP command to reclaim unused space in the data base. The system actually does not remove deleted members; it merely flags the space as unusable until you compress it. Then the system moves other members into that space and displays a message after each member it moves. When the system completes the compress, it displays the following message:

```
COMPRESS COMPLETED
```

You should exercise caution in using this function as it actually rearranges the members in the data base. To prevent unpredictable results, you should restrict your use of the interpreter ($DIINTR) during this process.

Note: If an unrecoverable I/O error occurs, it destroys the data set.

```
COMMAND (?):  CP
WARNING--COMPRESS IN PLACE.  IF AN ERROR
SHOULD OCCUR DATA SET WILL BE DESTROYED
  DO YOU WISH TO PROCEED?  Y
DATA         COPIED
RDATA        COPIED
REPORT       COPIED
SQUARE       COPIED
CIRCLE       COPIED
RPT          COPIED
ARC          COPIED
PLOT         COPIED
COMPRESS     COMPLETED

COMMAND (?):
```

CM — Copy Member

Use the CM command to copy a member from the source data base to the target data base. The options available under MD (move data base) are also available under CM (copy member).

```
COMMAND (?):  CM
SOURCE DATA SET NAME:  $DIFILE
LOCATED ON VOLUME:  EDX002
CHANGE SOURCE DATA SET?  N
TARGET (NAME, VOLUME):  $DIFILE,EDX003
SAVE EXISTING MEMBERS IN TARGET DATA BASE?  Y
ENTER MEMBER NAME TO BE COPIED
PLOT
PLOT         COPIED
COPY COMPLETED

COMMAND (?):
```

$DIUTIL - Maintain Partitioned Data Base *(continued)*

DE — Delete a Member

Use the DE command to remove display or data members from the data base. The system prompts you for the name of the member you want to delete and asks you to verify the accuracy of your entry prior to actual deletion.

```
COMMAND (?):    DE
MEMBER NAME:    PLTT
DELETE MEMBER PLTT?    Y
PLTT DELETED

COMMAND (?):
```

EN — Exit Program

Use the EN command to terminate the $DIUTIL utility.

IN — Initialize Data Base

Use the IN command to format the entire data base to zeros and to format the directory to reflect the starting and ending record numbers. The system prompts you to proceed.

Note: This function destroys any data in the data base.

Make sure you enter the correct data set name. IN ends when the system displays the message DATA SET FORMATTED. You allocated $DIFILE with $DISKUT1 Each directory record allocated with IN contains 16 directory entries, except the first, which contains 15.

```
COMMAND (?):    IN
*-*-WARNING THIS FUNCTION WILL DESTROY ANY DATA
                                CURRENTLY IN DATA SET-*-*

DO YOU WISH TO PROCEED?    Y
ENTER DIRECTORY SIZE IN RECORDS:    2
DATA SET FORMATTED
DATASET NAME: $DIFILE
LOCATED ON VOLUME: EDX002
- DATA SET -- DIRECTORY-
   NEXT    TOTAL   NEXT    TOTAL
      3      100      1       31

END OF STATUS

COMMAND (?):
```

$DIUTIL - Maintain Partitioned Data Base *(continued)*

LA — Display Directory

Use the LA command to display all active members. Each line of display shows the member name followed by four values:

1. Starting sector relative to the start of the data base.
2. Length of member in records.
3. Member usage code.
4. User-defined member code.

```
COMMAND (?):  LA
PLOT          11    4   2    0
DATA          15   10   5    0
RDATA         25   10   4    0
REPORT        35    1   1    0
SQUARE        36    1   2    0
CIRCLE        38    1   2    0
RPT           39    1   1    0
ARC           40    1   2    0

COMMAND (?):
```

LH — Display Member Header

Use the LH command to display the header of a data member (types 4-9). The header describes the characteristics and use of the member.

Example:

```
COMMAND (?):  LH
MEMBER NAME:  RDATA
MEMBER RDATA HEADER
      0     0      0      0      2     32     80      0
END OF HEADER

COMMAND (?):
```

$DIUTIL - Maintain Partitioned Data Base *(continued)*

MD — Move Data Base

Use the MD command to move the data base on the same or another volume when the data base becomes too small to add a member. You can move the online data base to another location temporarily, delete the old version, reallocate and initialize the new expanded version, and move back the previous contents. During this procedure, use the Interpreter with care.

Note: If you are moving the data base and the Interpreter uses a member, you will get unpredictable results.

During the execution of MD, the system prompts you for a new source data if you want one and a target data base. You have the option of saving the members in the target data base. MD is helpful if you wish to use $DICOMP to develop display members in a different data base than the online version and then, at a later time, combine the new members with those in the online data base.

```
COMMAND (?):   MD
SOURCE DATASET NAME:   $DIFILE
LOCATED ON VOLUME:   EDX002
CHANGE SOURCE DATASET?   N
TARGET (NAME,VOLUME):   $DIFILE,EDX003
SAVE EXISTING MEMBERS IN TARGET DATA BASE?   Y
PLOT         COPIED
DATA         COPIED
RDATA        COPIED
REPORT       COPIED
SQUARE       COPIED
CIRCLE       COPIED
RPT          COPIED
ARC          COPIED
COPY COMPLETED

COMMAND (?):
```

$DIUTIL - Maintain Partitioned Data Base *(continued)*

RE — Rename Member

Use the RE command to change the display profile ID name. The system prompts you for each step and takes no action until it obtains your response first. RE is useful when you need to modify an online member. You can copy the member that needs changing to another data base, modify and test it, then rename it and copy it back to the online data base. By using the rename and delete functions, you can exchange the new for the old without interfering with any online functions.

```
COMMAND (?):   RE
MEMBER NAME:   PLOT
ENTER NEW NAME:   PLTT
RENAME COMPLETED

COMMAND (?):
```

ST — Display Data Base Status

Use the ST command to display the current data base status. The first line shows the data base location and name. The data that follows is the current status of the data base. There are four values presented. The first is the next available record. The second is the total number of records in the data base. You can see then how much space is available for new members. If space is running short, you can compress the data base or allocate a larger area. The next value displayed is the next available directory entry. The last value displayed is the total number of directory entries available. Refer to these two values to determine if you need more or less space for directory entries. Following the completion of the status display, the system displays a message indicating end-of-status.

```
COMMAND (?):   ST
DATASET NAME:   $DIFILE
LOCATED ON VOLUME:   EDX002
- DATA SET -- DIRECTORY-
   NEXT    TOTAL   NEXT    TOTAL
     41      100     10      159

END OF STATUS

COMMAND (?):
```

$DUMP - Format and Display Saved Environment

$DUMP displays on a terminal or printer the contents of the data set generated by the $TRAP utility or stand-alone dump facility. After the successful execution of $TRAP and the subsequent occurrence of a trap condition, the data set assigned to $TRAP or the stand-alone dump will contain a storage image. Use $DUMP to retrieve, format, and print the data on a terminal or printer. The *Problem Determination Guide* shows how to interpret the output of $DUMP.

Notes:

1. To print the contents of a stand-alone dump or $TRAP diskette that you created with $DASDI, use the data set and volume name $$EDXLIB,IBMEDX.
2. Taking a stand-alone or $TRAP dump allows you to dump unmapped as well as mapped storage.
3. You can specify a partial dump of mapped storage but not of unmapped storage.

Invoking $DUMP

You invoke $DUMP with the $L command or option 9.1 of the session manager.

Example: Partial dump of partition 3 to printer using $TRAP output.

```
      > $L $DUMP
 1    DUMPDS(NAME,VOLUME):  DUMP,EDX003
     LOADING $DUMP    22P,12:20:17, LP=8F00, PART=1
 2    ENTER DEVICE NAME FOR OUTPUT:   $SYSPRTR
     USING $TRAP OUTPUT
 3   PARTIAL DISPLAY (Y/N)?   Y
 4   FORMAT CONTROL BLOCKS (Y/N)?   Y

 5   DUMP MAPPED STORAGE (Y/N)?   Y

 6   VALID PARTITIONS ARE
     1,  2,  3,  4,  5,  6,  7,  8,  9,  10,  11,  12,  13,  14,  15,  16
     ENTER PARTITIONS # :  3

 7   ENTER START AND ENDING ADDRESS IN HEX (2 WORDS):   400 500

 8   ANOTHER MAPPED AREA (Y/N)?   N

 9   DUMP UNMAPPED STORAGE (Y N)?   N

10   ANOTHER AREA (Y/N)?  N
```

$DUMP - Format and Display Saved Environment *(continued)*

1 The data set you specify here must be the same as the one you defined when you executed $TRAP.

2 You can specify a terminal to receive the output from $DUMP. If you press the ENTER key or enter $DUMP, the dump program assumes that it should direct the output to the terminal where you loaded $DUMP. Using the attention key followed by CA cancels any current $DUMP operation (such as control blocks or mapped storage,) but not the $DUMP program itself.

3 If you want a display all of storage, respond to this question with an N. If any unmapped storage resides in the data set, the system prompts you with DUMP UNMAPPED STORAGE?. After you answer the prompt, the output display begins immediately and continues until the system dumps all of storage or you enter an attention CA. If you respond Y, $DUMP allows you to display certain sections of mapped storage.

4 If you want a formatted display of the control blocks, respond to this question with a Y.

5 If you want to dump mapped storage, respond with a Y and you will get the PARTIAL DISPLAY? prompt. If you respond N, the system does not issue the PARTIAL DISPLAY? prompt.

6 Enter a number 1 through 16 for the valid partition number that contains the storage you want to dump.

7 Enter the starting and ending addresses of the storage you want to dump.

8 If you want to dump another mapped storage area, respond with a Y. The system then prompts for the valid partition.

9 If you want to dump unmapped storage, respond with a Y. The system then dumps all of unmapped storage.

10 $DUMP allows you to request several dumps. If you respond with a Y, $DUMP prompts you again, starting with the DUMP MAPPED STORAGE (Y/N)? prompt.

$DUMP - Format and Display Saved Environment *(continued)*

Example: Partial dump output of partition 1.

```
     EVENT DRIVEN EXECUTIVE FORMATTED STORAGE DUMP

     AT TIME OF TRAP PSW WAS 8002 ON HARDWARE LEVEL 2

               LEVEL 0   LEVEL 1   LEVEL 2   LEVEL 3

     IAR         0892      0892      0892      0892
     AKR         0000      0000      0000      0000
     LSR         00D0      0090      0090      0090
     R0          0000      0000      0000      0000
     R1          0000      0000      0000      0000
     R2          0000      0000      0000      0000
     R3          0000      0000      0000      0000
     R4          0000      0000      0000      0000
     R5          0000      0001      0002      0003
     R6          8000      8000      8000      8000
     R7          0000      0000      0000      0000

STORAGE SEGMENTATION REGISTERS:

BLOCK ADS0    ADS1    ADS2    ADS3    ADS4 ADS5 ADS6 ADS7

0000  0004    0104    0204    0304
0800  000C    010C    020C    030C
1000  0014    0114    0214    0314
1800  001C    011C    021C    031C
2000  0024    0124    0224    0324
2800  002C    012C    022C    032C
3000  0034    0134    0234    0334
3800  003C    013C    023C    033C
4000  0044    0144    0244    0344
4800  004C    014C    024C    034C
5000  0054    0154    0254    0354
5800  005C    015C    025C    035C
6000  0064    0164    0264    0364
6800  006C    016C    026C    036C
7000  0074    0174    0274    0374
7800  007C    017C    027C    037C
8000  0084    0184    0284    0384
8800  008C    018C    028C    038C
9000  0094    0194    0294    0394
9800  009C    019C    029C    039C
A000  00A4    01A4    02A4    03A4
A800  00AC    01AC    02AC    03AC
B000  00B4    01B4    02B4    03B4
B800  00BC    01BC    02BC    03BC
C000  00C4    01C4    02C4    03C4
C800  00CC    01CC    02CC    03CC
D000  00D4    01D4    02D4    03D4
D800  00DC    01DC    02DC    03DC
E000  00E4    01E4    02E4    03E4
E800  00EC    01EC    02EC    03EC
F000  00F4    01F4    02F4    03F4
F800  00FC    01FC    02FC    03FC
```

$DUMP - Format and Display Saved Environment *(continued)*

Partial dump output of partition 1 (continued).

```
STORAGE MAP:         $SYSCOM AT ADDRESS 1580

EDXFLAGS  5000      SVCFLAGS  8000

PART#     NAME       ADDR    PAGES     ATASK     TCBS(S)

 P1         ADS=0    0000     256
         TESTPGM     8000       7      8642(A)   85C2 83A0 82C4
         UTILITY     8700      21      9B58(A)   9AD8
         $PROGUT1    9C00      12                9E22
         **FREE**    9F00      97

 P2         ADS=1    0000     256
         **FREE**    2800     216

 P3         ADS=2    0000      80
         **FREE**    2800      16
         SAMPLE01    3800      20
         **FREE**    4C00       4
```

$DUMP - Format and Display Saved Environment *(continued)*

Partial dump output of partition 1 (continued).

```
     IO DEVICE DDB INFORMATION

   TERMINAL LIST:

     NAME       CCB  ID   IODA FEAT QCB  CUR-TCB   CHAIN

11 $SYSLOG     1466 0406 0004 0400 FFFF  NONE      NONE * ERROR-A
   $SYSPRTR    16A2 0306 0021 0020 0000 0000       9822-0  83A0-0
   $4978A      187C 040E 0024 0400 0000  4972-2    NONE
   CDRVTA      1A56 FFFF 0000 0800 FFFF  NONE      NONE
   CDRVTB      1C3A FFFF 0000 0000 FFFF  NONE      NONE

   DSK(ETTE)/TAPE VDE:
    VDE   NAME      DDB       FLAGS     QCB     CUR-TCB  CHAIN (TCB-ADS)

    077C  *DDE*     0834      0800      FFFF     NONE     NONE
    07AA  ASMLIB    0834      0000      0000     85C2-0   NONE
    07D8  EDX003    0834      0000      FFFF     NONE     NONE
    0806  *DDE*     091A      0800      FFFF     NONE     NONE
    08EC  BACKUP    0A00      0100      FFFF     NONE     NONE
    09D2  *OFFL*    0AE6      0000      FFFF     NONE     NONE

    DDB   IODA      DEVID     DSCB->    TASK DSCB-CHAIN
    0834  0003       00BA     8032-0    1166    3834-2*
    091A  0048       3106     0000-0    1166    3834-2
    0A00  0002       0106     0000-0    11E6    NONE
    0AE6  0012       0106     0000-0    1266    NONE

    PARTITION 1 BEGINNING AT ADDRESS 0000 FOR 139 PAGES

    0000  6802 882A 0000 0000 8968 8826 8696 8826 | ......
    0010  0000 0000 8968 8826 0A76 0A12 8968 8826 | ......
       SAME AS ABOVE
    0100  12DC 8BC2 0004 0006 1010 6A08 03F8 5B22 | ...B..
```

11 $DUMP detected several errors and listed them in the format portion of the output. They are:

*ERROR-A:	Invalid address
*ERROR-T:	Invalid TCB
*ERROR-L:	Reasonable chain limit (150) exceeded
*ERROR-D:	Address does not exist

$DUMP - Format and Display Saved Environment *(continued)*

If you have a program that is using unmapped storage, dump the partition where the program is running as well as the unmapped storage area. Determine which unmapped storage pointers relate to your program by locating the STORBLKS within your program. Then use the unmapped storage equate ($STORUSR) to determine the address of the start of the list of unmapped storage pointers within this STORBLK. The unmapped storage pointers listed within the STORBLK are owned by your program. They point to 2K blocks of storage which should have been printed out. Use these pointers to locate the correct 2K blocks in your $DUMP listing. Refer to the *Problem Determination Guide* for additional information.

The following examples show dumps using stand-alone dump output.

Example 1: How to print a stand-alone dump for mapped and unmapped storage

```
> $L $DUMP
 DUMPDS(NAME,VOLUME): $$EDXLIB,IBMEDX
LOADING $DUMP    22P,12:20:17, LP=8F00, PART=4

ENTER DEVICE NAME FOR OUTPUT: $SYSPRTR

USING STAND-ALONE DUMP OUTPUT
PARTIAL DISPLAY (Y/N)?  Y
FORMAT CONTROL BLOCKS (Y/N)? N

DUMP MAPPED STORAGE (Y/N)? Y
VALID PARTITIONS ARE:
1,  2,  3,  4,  5,  6,  7,  8,  9,  10,  11,  12,  13,  14,  15,  16
ENTER PARTITION # : 1
ENTER START ENDING ADDRESS IN HEX (2 WORDS): 0000 0800
ANOTHER MAPPED AREA (Y/N)?N

DUMP UNMAPPED STORAGE (Y/N)? Y

$DUMP REQUIRES DISKETTE #2, PLEASE MOUNT DISKETTE
PRESS "ENTER" TO CONTINUE OR "PF3" TO END DUMP
```

$DUMP - Format and Display Saved Environment *(continued)*

Example 2: How to copy a stand-alone dump from diskette(s) to disk.

```
  > $L $DUMP $$EDXLIB,IBMEDX
LOADING DUMP         77P,02:54:07, LP= 0800, PART= 3
ENTER DEVICE NAME FOR OUTPUT:   $SYSPRTR
PARTIAL DISPLAY (Y/N)?  Y
USING STAND ALONE DUMP OUTPUT

MAPPED STORAGE SPANS DISKETTE WHICH WILL CAUSE REPEATED
DISKETTE SWITCHING DURING CONTROL BLOCK FORMATTING

IF DUMP DATA SET IS COPIED TO DISK DATA SET, NO DISKETTE SWITCHING
DISK DATA SET MUST BE    8200 RECORDS IN LENGTH

COPY DISKETTE(S) TO DATA SET (Y/N)?  Y
TARGET (NAME,VOLUME):  DUMPDS,EDX003

$DUMP REQUIRES DISKETTE # 2, PLEASE MOUNT DISKETTE
PRESS "ENTER" TO CONTINUE OR "PF3" TO END DUMP

$DUMP REQUIRES DISKETTE # 3, PLEASE MOUNT DISKETTE
PRESS "ENTER" TO CONTINUE OR "PF3" TO END DUMP

$DUMP REQUIRES DISKETTE # 4, PLEASE MOUNT DISKETTE
PRESS "ENTER" TO CONTINUE OR "PF3" TO END DUMP

DUMP DATA SET COPIED TO DUMPS, EDX003

CONTINUE DUMP FROM DISK DATA SET (Y/N)?  Y
FORMAT CONTROL BLOCKS (Y/N)?  Y

VALID PARTITIONS ARE:
   1,  2,  3,  4,  5,  6,  7,  8,  9,  10,  11,  12,  13,  14,  15,  16
ENTER PARTITION # :  4
ENTER START AND ENDING ADDRESS IN HEX (2 VALUES): 200 600
ANOTHER MAPPED AREA (Y/N)?  N

DUMP UNMAPPED STORAGE (Y/N)?  N
ANOTHER AREA (Y/N)?  N

$DUMP    ENDED AT 03:46:33
```

$EDIT1 and EDIT1N - Line Editors

$EDIT1 and $EDIT1N provide a text editing facility (primarily used for source program entry and editing) that you can load while other programs are executing. The Host Communication Facility-related version ($EDIT1) provides a few commands for data communication using the Host Communications Facility IUP on the System/370; with them you can control almost the entire process of program preparation from a Series/1 terminal.

Both utilities work with 80-character lines with line numbers in positions 73-80. You invoke them with the $L operator command.

Data Set Requirements

The editing facility requires one work data set; you must allocate it on disk or diskette using $DISKUT1. The system prompts you for its name when you invoke either version. This data set contains both your data and some index information during the editing session. The size (number of records) of the data set determines the maximum number of data records that it can contain. It is divided into three parts:

1. One header record

2. A series of index records (32 entries per record)

3. A series of data records (3 entries per record).

You can calculate the required data set size as follows: number of text lines (n) divided by 30, times 11, plus 1 ((n/30 x 11) + 1).

Note: The data set must contain fewer than 32768 records.

Invoking $EDIT1 or $EDIT1N

You invoke $EDIT1 or $EDIT1N with the $L operator command. The session manager does not support either utility.

$EDIT1 and EDIT1N - Line Editors *(continued)*

Sequence of Operations

When you invoke $EDIT1 and $EDIT1N, they prompt you for the name of the work data set. If you are going to edit an existing data set, use the READ command to copy the data set to the work data set. For a new data set, enter edit mode. You can print the contents of the work data set by using the LIST command.

Use the EDIT command to enter edit mode. The system then recognizes "Edit Mode Subcommands" on page UT-247 until you end the utility with the END command.

Note: You should use the VERIFY ON subcommand until you become familiar with the editing process.

Use the TABSET subcommand if you want to specify the tab character and tab column. TABSET eliminates blanks when a substantial amount of the text you are entering is in tabular format or begins in a particular column.

You can enter data a line at a time under the INPUT subcommand (recommended for new data sets and bulk sequential updates because of the automatic prompting feature) or by using the line editing function (for single-line corrections). You can list portions of the edited data at the terminal by using the LIST command.

The FIND, TOP, BOTTOM, UP, and DOWN subcommands control the position of the current line pointer.

You can end edit mode with the END command. After you have edited the text, use either the WRITE or SAVE subcommand to copy the work data set to a permanent data set. The work data set is in a blocked format that is incompatible with most Event Driven Executive functions. Therefore, the system performs automatic translation from text editor format to source statement format.

The following figure shows the primary commands and subcommands available under $EDIT1/$EDIT1N.

$EDIT1 and EDIT1N - Line Editors *(continued)*

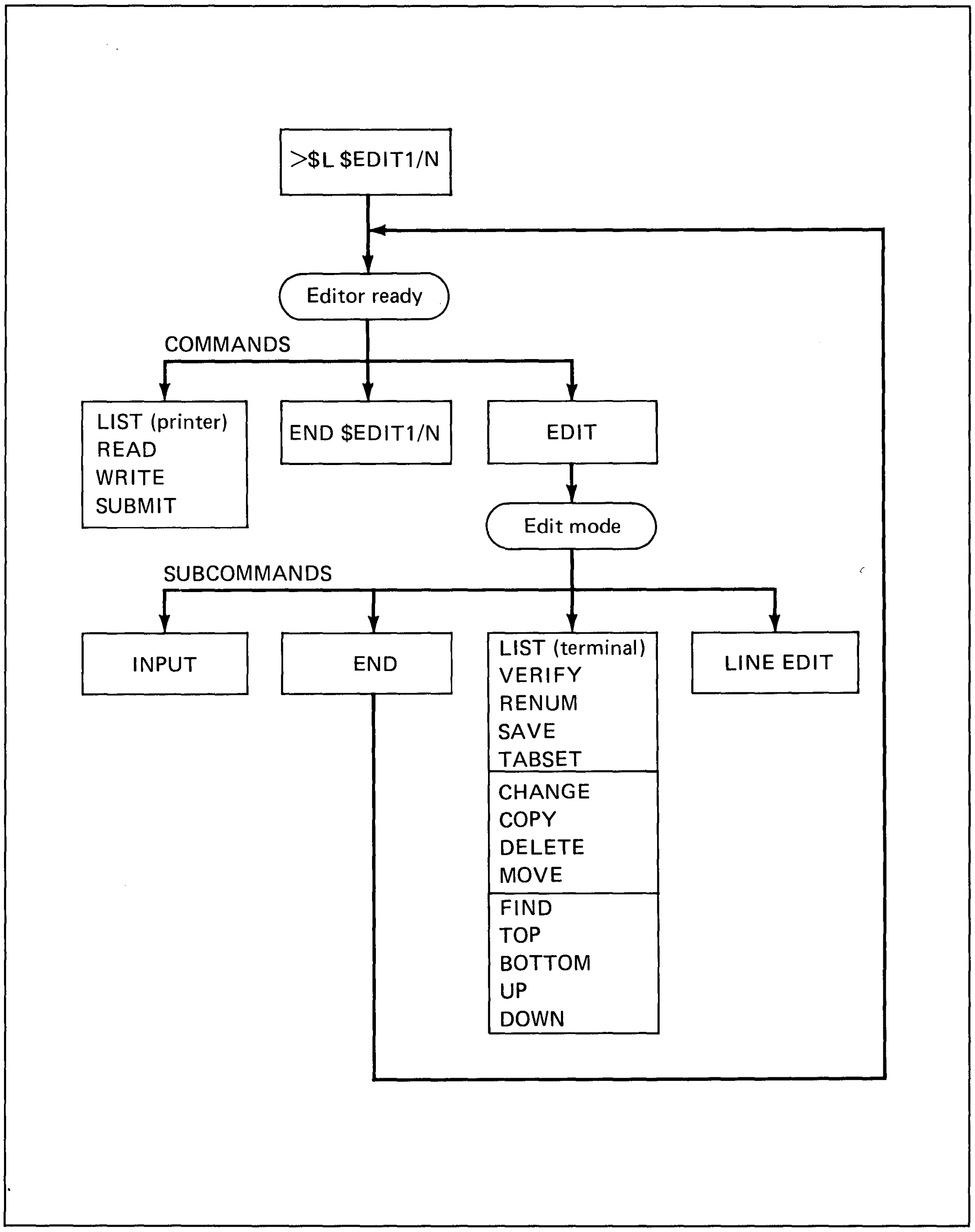


Figure 14. $EDIT1/$EDIT1N commands and subcommands

$EDIT1 and EDIT1N - Line Editors *(continued)*

Special Control Keys

1. End-of-Line Character (see note below). You can use the carriage return key (CR)/ENTER to end an input line.

2. Line-Delete Character (see note below). You can use the delete key (DEL) of certain teletypewriter terminals to delete an input line.

   Note: You can define the CR and DEL keys in the TERMINAL statement. See the *Installation and System Generation Guide.*

3. Character-Delete Character. You can use the backspace (BS) key on terminals for the character delete function. On teletypewriter terminals, use the CTRL and H keys simultaneously.

4. Tabulation Character. You can set the TAB character to the character of your choice. '%' is the default TAB character. Columns 10, 20, 40, and 72 are the default TAB columns.

5. ATTN Key (4978/4979), ALT MODE and PF8 Key (3101), or ESC or ALT MODE Key (teletypewriter terminals). You can cancel the subcommands CHANGE, FIND, and LIST, described below, by pressing the ATTN/ESC key and entering, as a special system utility function, the two-character code CA. This feature is useful, for example, to end a long listing.

Editor Commands

The editor commands are described in the following pages. Unless specifically indicated, the commands apply to both the host and native versions of this utility. The editor commands are:

COMMAND	DESCRIPTION
EDIT	Enters edit mode; allows edit subcommands
END	Ends $EDIT1 and $EDIT1N
LIST	Lists the work data set on the system printer
READ	Reads a source data set into the work data set
SUBMIT	Submits a job to the host batch job stream
WRITE	Writes the work data set into a source data set

The descriptions that follow show the syntax of the editor commands, including any operands associated with them.

$EDIT1 and EDIT1N - Line Editors *(continued)*

EDIT — Enter Edit Mode

Use EDIT to begin editing source data.

Syntax:

```
EDIT    OLD/NEW

 Required: none
 Defaults: NEW when using a newly allocated work data set
           OLD when using an old work data set
 Alias:    E,ED
```

Operands	*Description*
OLD	Indicates that data exists in the data set you want to modify.
NEW	Indicates that you are creating new data.

Notes:

1. You must enter the EDIT command before you can use the editor subcommands.
2. When you are in edit mode, you must enter the subcommand END or SAVE before you can use the editor commands listed on the preceding page.

$EDIT1 and EDIT1N - Line Editors *(continued)*

END — End $EDIT1/$EDIT1N

Use END to end execution of $EDIT1 or $EDIT1N.

The system will not change the contents of the edit work data set. You can reinvoke $EDIT1/$EDIT1N at a later time and continue.

Syntax:

```
END

 Required: none
 Defaults: none
 Alias:    EN
```

Operands ***Description***

None

$EDIT1 and EDIT1N - Line Editors *(continued)*

LIST — List Work Data Set

Use LIST to print all or part of the work data set on the system printer ($SYSPRTR). You can enter a single line number or a pair of line numbers to specify a line range. If you do not enter any line numbers, the system lists the entire data set. You can end the listing by pressing the attention key and entering CA. Note a similarity to the EDIT subcommand. If you use LIST as a command following READY, the the system prints the data set on $SYSPRTR. If you use LIST as a subcommand following EDIT, the system displays the data set on your terminal.

Syntax:

```
LIST    line-spec

 Required:  none
 Defaults:  none
 Alias:     L,LI
```

Operands	***Description***
line-spec	'*' (for the current line) or 'line-number' to indicate a single line to be listed. '* COUNT' or 'linenum1 linenum2' to display a range of lines. If you omit this operand, the system prints the entire data set.

Example:

```
LIST 10 100
L * 5
L *
LI
```

$EDIT1 and EDIT1N - Line Editors *(continued)*

READ — Retrieve Host Data Set ($EDIT1)

Use READ to retrieve a data set from the host system and store it in your Series/1 work data set.

You must have the Host Communications Facility on the System/370.

Syntax:

```
READ    dsname

 Required: none
 Defaults: If you omit dsname, the system prompts you
 Alias:    none
```

Operands	*Description*
dsname	The fully qualified name of the host data set to be retrieved. It must contain fixed length, 80-byte records, with line numbers in columns 73-80. You can enter the command and name together on the same line or enter the command READ and the system prompts you for the data set name.

$EDIT1 and EDIT1N - Line Editors *(continued)*

READ — Retrieve Series/1 Data Set ($EDIT1N)

Use READ to retrieve a named data set from a volume on the Series/1 disk or diskette and store it in a Series/1 work data set.

Syntax:

```
READ      dsname volname

 Required:  none - system prompts for operands
 Default:  none
 Alias:    R, RE
```

Operands	*Description*
dsname	Name of data set you want to retrieve
volname	Name of the volume containing the data set you want to retrieve

Note: You may enter these operands as responses to system prompts.

$EDIT1 and EDIT1N - Line Editors *(continued)*

SUBMIT — Submit Job to Host ($EDIT1)

Use SUBMIT to inject a job (job control statements and optional data) into the host batch job stream.

You must have the Host Communications Facility on the System/370.

Note: Use this option only in systems with a HASP or JES/Host Communication Facility interface.

Syntax:

```
SUBMIT   dsname|DIRECT
```

Operands	*Description*
dsname	The fully qualified name of the host data set, the contents of which you want entered into the job stream. This data set must contain fixed length, 80-byte records.
DIRECT	If you specify it, the system transfers the contents of your edit work data set directly to the host job stream.

$EDIT1 and EDIT1N - Line Editors *(continued)*

WRITE — Write Work Data Set to Host ($EDIT1)

Use WRITE to transfer your Series/1 work data set to a host data set. The system assumes that you have edited or created your data set with the $EDIT1 utility program.

You must have the Host Communications Facility on the System/370.

If you previously specified a host data set, the utility asks if you wish to reuse it. If you do not, or if you did not specify one previously, the utility prompts you for a new host data set name.

Syntax:

```
WRITE   dsname
```

Operands	*Description*
dsname	The fully qualified name of the target host data set. This data set should contain fixed-length, 80-byte records. You can enter the command and name together on the same line or enter only the command WRITE. The system prompts you for the data set name.

EDIT1 issues the following prompt:

```
NAME:
```

$EDIT1 and EDIT1N - Line Editors *(continued)*

WRITE — Write Work Data Set to Series/1 Data Set ($EDIT1N)

Use WRITE to copy the Series/1 work data set to a named data set in a Series/1 disk or diskette volume.

Syntax:

```
WRITE

Required:  none - System prompts you for operands
Default:   copy to the originating data set, if any
Alias:     W, WR
```

Operands ***Description***

None

EDIT1N issues the following prompt:

```
WRITE TO READVS ON READVOL (Y/N)?
```

where READVOL is the originating volume and READVS is the originating data set. The system issues this prompt only if you initialized the work data set via the READ command. If you respond NO or if the data set is new, the system issues the following prompt:

```
ENTER (NAME,VOLUME):
```

$EDIT1 and EDIT1N - Line Editors *(continued)*

Edit Mode Subcommands

The subcommands used to edit your work data set while in EDIT mode are described as follows:

Subcommand	Operands
BOTTOM	
CHANGE	line-spec /text1/text2/ALL
COPY	line-spec
DELETE	line-spec
DOWN	count
END	
FIND	/character-string/
INPUT	line-number increment
Line Editing	line-number character-string
LIST	line-spec
MOVE	line-spec
RENUM	new-line-number increment
SAVE	
TABSET	ON(integer list), OFF, CH(character)
TOP	
UP	count
VERIFY	ON/OFF

The descriptions that follow show the syntax of the subcommands including any operands associated with them.

BOTTOM — Set Line Pointer to Bottom

Use BOTTOM to reposition the current line pointer (*) to the last line of the data set you are editing.

Syntax:

```
BOTTOM

 Required: none
 Defaults: none
 Alias:    B,BO
```

Operands ***Description***

None

$EDIT1 and EDIT1N - Line Editors *(continued)*

CHANGE — Change Character String

Use CHANGE to modify a character string in a line or range of lines.

Syntax:

```
 CHANGE   line-spec /text1/text2/ALL

Required: /text1/text2
Defaults: line-spec defaults to *.
Alias:    C,CH
```

Operands	***Description***
line-spec	* or blank for the current line. '* count' or 'linenum1 linenum2' for a range of lines. 'line-number' for a particular line.
/text1/text2/ALL	'/' can be any nonnumeric character except blank, tab, and asterisk. It is not a part of and cannot appear within the character strings 'text1' and 'text2'. The system searches the line or range of lines for 'text1', which it replaces with 'text2' if it finds it. Note that you must use the same character for both delimiters in any one change command. The keyword 'ALL' is optional and causes the system to replace every occurrence of 'text1' in the line(s). Two adjoining delimiters denote a null operand. If text1 is a null operand, then the system inserts text2 at the start of the line and shifts the line to the right. If text2 is a null operand and you specify text1, the system removes text2 from the line and shifts the rest of the line to the left.

Example:

```
C 20 /ABC/ADC/
C 100 250 =/*=//=ALL
C * //XYZ
C /PROG/PGM/
```

$EDIT1 and EDIT1N - Line Editors *(continued)*

COPY — Copy Text

Use COPY to duplicate text from one location in a data set to another location within that data set. The 'from' and 'to' text both remain in the data set.

Syntax:

```
COPY     linenum1 linenum2 linenum3

 Required: linenum1 linenum3
 Defaults: linenum1 linenum3 defaults to
         a single line copy of '1' to '3'.
 Alias:   CO
```

Operands	***Description***
linenum1	The first line of text you want to copy.
linenum2	The last line of text you want to copy.
linenum3	The line of text after which you want to place the copied text.

All specified line numbers must exist. 'linenum2' must be equal to or greater than 'linenum1'.

'linenum3' must be less than 'linenum1' or equal to or greater than 'linenum2' when you specify three line numbers.

The system renumbers the data set with standard specifications. It lists the original 'linenum2' with its new line number on exit.

Examples:

```
CO 100 300 60
CO 120 250 820
CO 150 150 310
COPY   150 310
```

Note: The last two examples are equivalent.

$EDIT1 and EDIT1N - Line Editors *(continued)*

DELETE — Delete Text

Use DELETE to remove records from the data set. The system repositions the current line pointer (*) prior to the deleted lines.

Syntax:

```
 DELETE   line-spec

Required: none
Defaults: *
Alias:    DE
```

Operands	*Description*
line-spec	* for current line.
	'* count' or 'linenum1 linenum2' for a range of lines.
	'line-number' for a particular line.

Examples:

```
DELETE *
DE * 4
DE 100 150
DE 125
```

$EDIT1 and EDIT1N - Line Editors *(continued)*

DOWN — Move Line Pointer Down

Use DOWN to move the current line pointer (*) toward the end of the data set.

Syntax:

```
DOWN     count

 Required: none
 Defaults: 'COUNT' defaults to 1.
 Alias:    DO
```

Operands	*Description*
count	Specifies the number of lines you want to move the current line pointer.

Examples:

```
DOWN 5
DO 10
```

$EDIT1 and EDIT1N - Line Editors *(continued)*

END — Exit Edit Mode

Use END to request that the system end the EDIT mode. You can use the editor commands now relative to your finished source data. To save or list your data set or to write or submit your data set to the host, see "Editor Commands" on page UT-238. The contents of the work data set remain unchanged. You can reenter the edit mode using the EDIT command and continue editing the work data set.

Syntax:

```
 END

Required: none
Defaults: none
Alias:    EN
```

Operands	*Description*
None	

$EDIT1 and EDIT1N - Line Editors *(continued)*

FIND — Find Character String

Use FIND to search for a specific character string beginning with the current line, if you specified operands. The system moves the current line pointer (*) to the first line it finds that contains the string. It searches every position within each line.

Note: You should set VERIFY to ON when you use the FIND command.

Syntax:

```
FIND     =char-string=

Required: none
Defaults: If no operands are specified, those specified
          on the last previous issue of the FIND
          subcommand are assumed. The search begins at
          the line following the current line.
Alias:    F,FI
```

Operands	*Description*
=char-string=	You can choose any nonnumeric character which does not appear within the specified character string (except BLANK, TAB, or ASTERISK) to be the string delimiter. You can replace the second occurrence with a carriage return. Note that both delimiters must be the same character.

Examples:

```
FIND /START/
F
FI =DATA  X'00F1'=
```

$EDIT1 and EDIT1N - Line Editors *(continued)*

INPUT — Input Text

Use INPUT to add or replace lines. You can use INPUT any time in edit mode by pressing the enter key. The system then adds lines to the end of the data set.

To end INPUT mode, press the enter key immediately after you receive the prompt for the next line number.

Syntax:

```
INPUT    line-number increment
             or
           * increment

 Required:  none
 Defaults:  increment defaults to previous one or to 10
 Alias:    I,IN
```

Operands	***Description***
line-number	The first line the system inserts will have this number or this number plus the increment if the specified line number already exists.
increment	The increment for numbering inserted lines; the default is the increment you specified previously or 10 if you don't specify another.
*	The system inserts lines at the current line position plus the default increment; if you specified no operands, the system inserts lines at the end of the data set plus the default increment.

Examples:

```
INPUT * 1
IN 100 5
I 20
I
```

$EDIT1 and EDIT1N - Line Editors *(continued)*

LIST — List Work Data Set

Use LIST to display, at the terminal, lines of the data set you are editing.

Syntax:

```
LIST     line-spec

 Required: none
 Defaults: line-spec defaults to entire data set
 Alias:    L,LI
```

Operands	*Description*
line-spec	(*) or line-number to indicate a single line you want to list; '* count' or 'linenum1 linenum2' to display a range of lines.

Examples:

```
LIST 10 100
L * 5
L *
LI
```

$EDIT1 and EDIT1N - Line Editors *(continued)*

MOVE — Move Text

Use MOVE to move text from one location in a data set to another location within that data set. The system deletes the 'from' text and leaves only the 'to' text in the data set.

Syntax:

```
MOVE    linenum1 linenum2 linenum3

 Required: linenum1 linenum3
 Defaults: linenum1 linenum3 defaults to move one line.
 Alias:    MO
```

Operands	***Description***
linenum1	The first line of text you want to move.
linenum2	The last line of text you want to move.
linenum3	The line of text after which you want to place the moved text.

All specified line numbers must exist.

'linenum2' must be equal to or greater than 'linenum1'.

'linenum3' must be less than 'linenum1' when you specify two line members or greater than 'linenum2' when you specify three line members.

The system renumbers the data set with standard specifications and lists the original 'linenum2' with its new line number on exit.

Examples:

```
MO 100 300 60
MO 120 250 820
MO 87 87 310
MOVE 87 310
```

Note: The last two examples are equivalent.

$EDIT1 and EDIT1N - Line Editors *(continued)*

RENUM — Renumber Work Data Set

Use RENUM to renumber each line of a line-numbered data set or to assign line numbers to each line of an unnumbered data set.

Syntax:

```
RENUM    new-line-number increment

 Required: none
 Defaults: Both new-line-number and increment
        default to 10
 Alias:   R, RE
```

Note: 'new-line-number' is required if 'increment' is specified.

Operands	*Description*
new-line-number	The sequence number you want to assign to the first line processed.
increment	The increment you want to use in renumbering.

Examples:

```
RENUM 10 10
RE 100 5
RENUM
R
```

$EDIT1 and EDIT1N - Line Editors *(continued)*

SAVE — Save Work Data Set

Use SAVE to write the current contents of the work data set to a host data set with the host-related version ($EDIT1) or to a Series/1 data set with the native-related version ($EDIT1N).

If you specified a data set previously (e.g., in a READ command), the system asks you if you wish to write onto that data set; otherwise, it prompts you for a new data set name.

Syntax:

```
SAVE     dsname

 Required: none
 Defaults: none
 Alias:    S, SA
```

Operands	***Description***
dsname	When you use $EDIT1, the system prompts you for the target host data set name; it must be a fully qualified data set name. When you use $EDIT1N, you must have allocated the target data set previously in a volume on a Series/1 disk or diskette. The data set should contain fixed-length, 80-byte records. The system prompts you for the target-volume name.

Examples:

```
SA
S
SAVE
```

$EDIT1 and EDIT1N - Line Editors *(continued)*

TABSET — Set Tabs

Use TABSET to reestablish tab values or nullify existing tab values. The system maintains the tabulation character and tab stop values as part of your work data set. (You can change them later.)

You can enter the tab character anywhere in the data line under the INPUT subcommand or line editing function. It causes a skip to the next tab position if you enter the data line into the work data set. The resulting line is not visible, but you can display it if you want.

Syntax:

```
TABSET   ON(integer-list)
TABSET   OFF
TABSET   CH(tab-character)

 Required: ON, OFF, or CH
 Defaults: none
 Alias:    TA
```

Operands	***Description***
integer-list	The relative column positions in each line to which you want the tab values set; initial system defaults are 10, 20, 40, and 72.
tab-character	A new tab character; the standard is a percent sign, %.
OFF	Resets relative tab column positions to initial systems defaults. It does not reset tab characters.

$EDIT1 and EDIT1N - Line Editors *(continued)*

Example 1: These lines show the various ways of entering TABSET subcommands and parameters.

1. The system sets the relative tab column positions to initial system defaults of 10, 20, 40, and 72.

```
TABSET ON(10 20 40 72)
```

2. The system sets the relative tab column positions to 10, 16, and 31.

```
TA ON(10 16 31)
```

3. The system designates the tab character as the pound sign, #.

```
TA CH(#)
```

4. The system resets the relative tab column values to the initial system defaults.

```
TA OFF
```

Example 2: The lines below show how you can set tab positions on different lines and then verify them in the EDIT mode.

1. The system sets the tab column positions to 10 and 20.

```
TABSET ON(10 20)
```

2. The system sets tab position 1 on line 36 of the work data set. The tab character is the default, the percentage sign, %.

```
36 %TAB POSITION 1
```

3. Use the INPUT subcommand to add two lines to the work data set: lines 37 and 38.

$EDIT1 and EDIT1N - Line Editors *(continued)*

```
INPUT 37 1
```

4. The INPUT subcommand prompt asks for further entries in the input mode. If you don't want any more entries, exit the input mode pressing the enter key.

```
INPUT
```

5. The system displays line 37 and sets tab position 2 using the tab character, %.

```
00037 %%TAB POSITION 2
```

6. The system displays the new line 38.

```
00038
```

7. Enter the EDIT command to get back into the edit mode.

```
EDIT
```

8. Use the LIST subcommand to request the display of lines 36 and 37 of the work data set.

```
LIST 36 37
```

9. The system displays data set lines 36 and 37, showing where it placed tab positions 1 and 2 within the lines.

```
00036          TAB POSITION 1
00037                    TAB POSITION 2
```

$EDIT1 and EDIT1N - Line Editors *(continued)*

TOP — Set Line Pointer to Top

Use TOP to position the current line pointer (*) before the first line of the data set.

Syntax:

```
  TOP

Required: none
Defaults: none
Alias:   TO
```

Operands	*Description*
None	

Note: If VERIFY is ON, the system prints no line because the current line number precedes the first line.

UP — Move Line Pointer Up

Use UP to move the current line pointer (*) toward the start of the data set.

Syntax:

```
  UP      count

Required:  none
Defaults:  count defaults to 1
Alias:    U
```

Operands	*Description*
count	The number of lines that you want the current line pointer (*) moved.

Example:

```
UP 10
```

$EDIT1 and EDIT1N - Line Editors *(continued)*

VERIFY — Display Changes on Terminal

Use VERIFY to display the changes you made on the terminal (ON); verification is OFF until you invoke it the first time during an edit.

Syntax:

```
VERIFY   ON/OFF

 Required: none
 Defaults: ON
 Alias:    V,VE
```

Operands	***Description***
ON	Each time the position of the current line pointer (*) changes, the system prints the line to which it moves. In addition, the system verifies modifications you made in fields of records using the 'character-string' or 'text' forms of the CHANGE subcommand.
OFF	The system will not verify changes you make to the position of the current line pointer (*) and to fields of records by means of the CHANGE subcommand.

Examples:

```
V ON
V
V OFF
VERIFY
```

$EDIT1 and EDIT1N - Line Editors *(continued)*

Line Editing Commands

The line editing commands allow you to add, replace, or delete a single line from the data set you are editing.

Note: Line editing functions are not subcommands.

Syntax:

```
 line-number character-string

Required: line-number
Defaults: none
```

Note: If you specify 'character-string', you must separate it from 'line-number' by a single blank or tab.

Operands	***Description***
line-number	'line-number' with no character string deletes the line having the specified number (it does nothing if the line does not exist). 'line-number', followed by a character string, adds the string to the data set. If a line having the specified number already exists, the system replaces it.
character-string	The text of the line you want to add.

Line Editing Examples: Add a line (line #12345 does not exist).

```
12345 This line is being added
```

Delete line 12345.

```
12345
```

Replace line 12345.

```
12345 This line replaces 12345
```

$EDXASM - Event Driven Language Compiler

The Event Driven Language compiler, $EDXASM, translates source programs coded in the Event Driven Language into object modules. Multiple copies of $EDXASM can operate concurrently if each copy has its own separate data sets.

Before using $EDXASM, you must enter the source program you want compiled onto a disk, diskette, or tape data set by means of one of the text editor utilities ($EDIT1N or $FSEDIT).

Required Data Sets

The $EDXASM compiler requires four data sets. You must specify the first three data sets in the order shown when you use the $L operator command to invoke $EDXASM. You must specify data sets one and three when you invoke $EDXASM using the session manager. You do not need to specify the name of the fourth data set. The system uses the default name $EDXL unless you change it at run time using the CONTROL (CO) option.

1. The *source program input data set (DS1)* contains the program you want to compile. Use a text editor ($EDIT1N or $FSEDIT) to create the source data set. You must specify the name of this data set when you invoke $EDXASM.

2. The compiler uses the *work data set (DS2)*. It contains object code, relocation pointers, and the symbol table. You must allocate this data set if you use the $L operator command to invoke $EDXASM. Allocate 100 records for a small program, 250 records for an average-sized program, and 500 records for a large program. Specify the name of the data set you allocated as a parameter on the $L operator command. The session manager automatically allocates this data set with a size of 400 records.

3. The *object data set (DS3)* will contain the output object module. It is input to $EDXLINK. The size is dependent on the program size. For estimation purposes, divide the program length in bytes by 100 to get the number of records required. In most cases, 25 to 50 records should be sufficient. You must allocate this data set and specify its name when you invoke $EDXASM. The EOD pointer is set to the end of the data after the system writes the output object module.

4. The *language control data set (DS4)* contains control information that $EDXASM uses. This data, named $EDXL, is divided into two logical parts: the error messages and the operation codes to process module specifications. The format of this data set enables you to modify the data set by using either text editing utility ($EDIT1N or $FSEDIT).

   $EDXL in volume ASMLIB contains the standard compiler error messages and Event Driven Language instruction set specifications. You may wish to add COPY code definitions or additional processing modules and error messages and may even desire to have differently modified versions assigned to different users. Use the *COPYCOD function to add additional modules to $EDXL. The contents of $EDXL are described in the *Internal Design*.

   To conserve space and speed, $EDXASM does not always flag operands as errors when the operand does not belong to the instruction. If not flagged, the erroneous operand does not expand or affect the instruction.

$EDXASM - Event Driven Language Compiler *(continued)*

Invoking $EDXASM

You invoke $EDXASM with the $L operator command, by option 2.1 of the session manager, or by the $JOBUTIL utility. In each case, you must provide the same information when you load $EDXASM for execution.

You can cancel the compilation or the subsequent listing at any time by pressing the attention key and entering CA.

Compiler Options

You can specify the option(s) you want when you invoke $EDXASM. The options you select direct the processing that the compiler will do.

Each option name is followed by its two-character abbreviation.

Basic Options

The following basic compiler options are available:

CANCEL (CA)	Causes the immediate termination of $EDXASM.
END (EN)	Ends option selection.
ERRORS (ER)	Specifies that the system should print only statements with errors. The system requires a device name. Similarly, a null entry or asterisk (*) indicates that you want the system to list the error messages on the loading terminal. Use this option for the first few compilations to remove typographical or simple syntactical errors from the source program.
LIST (LI)	Indicates that you want the system to print a full program listing (this is the default). Specify LIST only if you want the listing to go to a printer other than $SYSPRTR (the default).
NOLIST (NO)	Indicates that you don't want a listing. The system prints the compiler statistics on the terminal where you loaded $EDXASM.

$EDXASM - Event Driven Language Compiler *(continued)*

Special Options

The following special options are available:

BUILD (BU)	Causes the system to construct and write $EDXASM disk location information and instruction set table to the language-control data set. Use BUILD only when the language-control data set has changed or when the disk location of one or more parts of $EDXASM has changed. If a load fails because of outdated disk directory information, the system prints an error message and ends the assembly.
CONTROL (CO)	Specifies the language-control data set you want to use. If the system requires a language-control data set other than $EDXL on the volume from which $EDXASM was loaded, enter CONTROL followed by the name and volume of the data set you want to use.
OVERLAY (OV)	Specifies the number of storage areas into which $EDXASM is loaded. The default is six, with a minimum of one and maximum of six. A large number of storage areas reduces the number of storage loads the system requires, thus improving performance. If sufficient storage is not available for the requested number of storage areas, $EDXASM allocates the number of storage areas that do fit into the available space. You can find further information on multiple overlay area management in the $EDXASM section in the *Internal Design*.
RESET (RE)	Causes the system to clear the area in the language-control data set where BUILD stored overlay and instruction information. If this overlay and instruction information is present in the language-control data set, $EDXASM uses it instead of reconstructing the information for each execution, with a substantial savings in execution time.

Data Sets Used in Examples

The following data sets are used in the examples:

ASMSRC	The source input data set.
ASMWORK	The work data set (the session manager supplies a different work data set).
ASMOBJ	The object output data set.

If you do not specify the volume name for these data sets, the system uses the IPL volume as the default volume.

$EDXASM - Event Driven Language Compiler *(continued)*

Compiling a Program Using the $L Command

The following examples show how to invoke $EDXASM using the $L operator command. Both prompt/reply mode and single-line entry of commands are shown. For these examples, $EDXASM is stored on ASMLIB.

For examples of compiling a program using the session manager or the job stream processor utility ($JOBUTIL), see the *Event Driven Executive Language Programming Guide.*

Specifying Data Sets After Invoking $EDXASM

After you have entered the $L operator command, $EDXASM prompts you for the names of the required data sets.

```
 > $L $EDXASM,ASMLIB
 SOURCE  (NAME,VOLUME):    ASMSRC
 WORKFILE(NAME,VOLUME):    ASMWORK
 OBJECT  (NAME,VOLUME):    ASMOBJ
LOADING $EDXASM      64P,02:48:50, LP= 6300, PART=1
```

The following single-line entry is equivalent to the multiline entries above:

```
 > $L $EDXASM,ASMLIB ASMSRC ASMWORK ASMOBJ
```

After you enter the names of the required data sets, $EDXASM prompts you for a compiler option.

```
SELECT OPTIONS (?):
```

$EDXASM - Event Driven Language Compiler *(continued)*

Selecting Compiler Options

To display a list of the compiler options, enter a question mark in response to the SELECT OPTIONS prompting message as follows:

```
SELECT OPTIONS (?):  ?

LIST      -  SPECIFY LIST DEVICE
NOLIST    -  DO NOT PRINT LISTING
ERRORS    -  LIST ERRORS ONLY
CONTROL   -  SPECIFY CONTROL LANGUAGE
BUILD     -  BUILD OPCODE TABLE
RESET     -  RESET OPCODE TABLE
OVERLAY   -  SPECIFY NO. OF OVERLAY AREAS
CA        -  CANCEL ASSEMBLY
END       -  END OPTION SELECTION
 ('ATTN - CA' TO CANCEL ASSEMBLY DURING EXECUTION)

SELECT OPTIONS (?):
```

Press the ENTER key to let the compiler options default.

You can enter all option entries on a single line or in response to prompt messages. The last listing option you enter takes precedence.

The following examples show the use of various option selections.

Print Full Compiler Listing on $SYSPRTR (Default): If you don't select select any options (indicated by entering only a carriage return or ENTER in response to the select option message), the default is LIST on $SYSPRTR using the language control data set $EDXL on the $EDXASM program volume (ASMLIB).

```
SELECT OPTIONS (?):    (null entry)
```

Print Compiler Errors Only on Printer Named PRINTER1: If you want a compiler listing on another device, specify LIST or L. Enter the name of the device in response to the prompt for device name or enter it all on the same line. Use a null entry or an * to specify the terminal you are using.

$EDXASM - Event Driven Language Compiler *(continued)*

Both of the following examples print errors only on PRINTER1.

```
SELECT OPTIONS (?):  ERRORS
 DEVICE NAME:  PRINTER1

SELECT OPTIONS (?):  END
```

or

```
SELECT OPTIONS (?):  ER PRINTER1 END
```

Print No Listing and Use Control Data Set $EDXL on EDX002: Specify the name of the control-language data set you want to use by selecting the CONTROL option. Use this option when the control data set is different than the default.

The following example prints no compiler listing and uses the control data set $EDXL on volume EDX002.

```
SELECT OPTIONS (?):  NO CONTROL
CONTROL (DSNAME,VOLUME):  $EDXL,EDX002
SELECT OPTIONS (?):  END
```

Output of the Compiler

This section describes the output of the $EDXASM compiler:

- Statistics
- Object module
- Program listing
- Completion codes

$EDXASM - Event Driven Language Compiler *(continued)*

Statistics

When the system completes the compilation process, the compiler prints statistics indicating:

- Source, work, and object data sets used
- Date and time the compilation started
- Elapsed time for the compilation
- Number of statements processed
- Number of statements flagged with error messages.

Object Module

The system stores the object module at the end of the compilation. Then it is ready for input to $EDXLINK.

Since the system stores the output object module before it starts any listing, $EDXLINK can process the object module while the system is producing the listing. The operation of $EDXLINK is described in "$EDXLINK - Linkage Editor" on page UT-274.

Note: During assembly, there is a possibility that the system may have given the same value to different labels, causing assembly errors. If this occurs, modify one of the labels and reassemble the source.

Program Listing

If you request a program listing, the system prints it on the appropriate output device. The listing routine of $EDXASM automatically suppresses the printing of duplicate lines of object code. Before the system uses the data sets specified in the compilation, you can request a listing of the compilation using the program $EDXLIST. Refer to "Obtaining Extra Compilation Listings" for more information.

Completion Codes

The system prints completion codes on the invoking terminal and on a printer device. Refer to *Messages and Codes* for $EDXASM completion codes.

Obtaining Extra Compilation Listings

Use $EDXLIST to obtain a listing of the last $EDXASM compilation that the system performed.

$EDXASM - Event Driven Language Compiler *(continued)*

Invoking $EDXLIST Automatically

You can specify LIST or NOLIST in response to the SELECT OPTIONS (?): prompt when you load $EDXASM. A response of the enter key, LIST, or ERRORS, causes $EDXASM to load $EDXLIST and produce a listing of the compilation. If you respond with NOLIST, the system displays statistics from the compilation on the loading terminal and produces no listing.

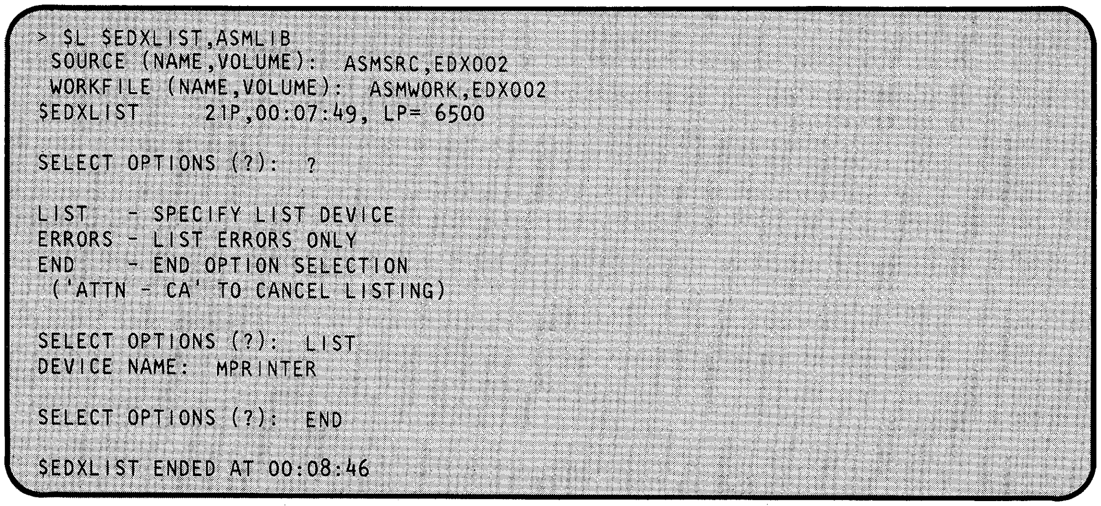
Invoking $EDXLIST Manually with the $L Operator Command

You can load $EDXLIST as a separate program. For example, if you select NOLIST and the statistics displayed at the end of the compilation indicate compilation errors, you can load $EDXLIST to print a listing.

You invoke $EDXLIST with the $L operator command. It requires two data set names: the source data set and work data set used for compilation. If you invoked $EDXASM with the session manager, the work data set is named $SM1user (where user is the sign-on ID).

Note: If you want a listing of the latest compilation, use $EDXLIST before you invoke $EDXASM again. Any subsequent compilation modifies the contents of the work data set.

An example of using $EDXLIST follows:



```
> $L $EDXLIST,ASMLIB
 SOURCE (NAME,VOLUME):  ASMSRC,EDX002
 WORKFILE (NAME,VOLUME):  ASMWORK,EDX002
$EDXLIST     21P,00:07:49, LP= 6500

SELECT OPTIONS (?):  ?

LIST   - SPECIFY LIST DEVICE
ERRORS - LIST ERRORS ONLY
END    - END OPTION SELECTION
 ('ATTN - CA' TO CANCEL LISTING)

SELECT OPTIONS (?):  LIST
DEVICE NAME:  MPRINTER

SELECT OPTIONS (?):  END

$EDXLIST ENDED AT 00:08:46
```

Figure 15. $EDXLIST use example

$EDXASM - Event Driven Language Compiler *(continued)*

Invoking $EDXLIST with $JOBUTIL

You can invoke $EDXLIST using the job stream processor $JOBUTIL The same options are available through the parm facility of $JOBUTIL as were described under "Invoking $EDXLIST Manually with the $L Operator Command" on page UT-272.

Example: Sample $JOBUTIL procedure of invoking $EDXLIST

```
PROGRAM  $EDXLIST,ASMLIB
NOMSG
PARM     LIST $SYSPRTR END
DS       ASMSRC,EDX002
DS       ASMWORK,EDX002
EXEC
```

$EDXLINK - Linkage Editor

The $EDXLINK program is a linkage editor that prepares programs to execute in an EDX system. Using $EDXLINK, you can format one or more separately assembled object modules into a nonrelocatable EDX supervisor or a relocatable load module.

With $EDXLINK you can:

- Produce a formatted storage map of the combined object modules.
- Include required routines from specified data sets.
- Define a single-level overlay structure for more efficient utilization of main storage.
- Define the number of overlay segments to execute in unmapped storage.
- Use interactive or noninteractive sessions.
- Run multiple links within the same session.
- Run from a terminal using the session manager or $L, or through $JOBUTIL.
- Log all error and warning messages to your terminal and to your specified output device.
- Specify additional dynamic storage to improve the performance of $EDXLINK.
- Use GLOBALS as references to $SYSCOM.

You can create object modules from any of the following EDX language processors to be input to $EDXLINK.

- The Event Driven Language compiler ($EDXASM)
- The Series/1 assembler ($S1ASM)
- The FORTRAN compiler
- The COBOL compiler
- The Pascal compiler
- The PL/I compiler
- Series/1 host assembler

$EDXLINK - Linkage Editor *(continued)*

Required Data Sets

$EDXLINK requires one work data set (DS1). Used for symbol resolution, the data set must contain at least 256 records. Two types of errors can occur when using this data set, I/O errors and an end-of-file (EOF). If an I/O error occurs, reallocate the data set to a different location on disk. If an end-of-file (EOF) occurs, the data set is too small. Reallocate the data set specifying a larger size.

Note: You must restart the link process if either of these errors occurs.

Optional data sets you may use when using $EDXLINK in a noninteractive mode are the *primary-control-statement data set* and the *secondary-control-statement data set.*

- The *primary-control-statement data set* contains the control statements $EDXLINK will execute when running in noninteractive mode. It can contain secondary-control-statement data sets if you use the COPY control statement.

- The *secondary-control-statement data set* contains additional control statements that the primary-control-statement data set will use. These are common control statements that are used frequently for many different link edits.

$EDXLINK Control Statements

Control statements are the instructions used by $EDXLINK to convert separately compiled or assembled object module(s) into an executable load module. You enter the control statements one at a time in interactive mode or write the entire set of link control statements to a link-control data set for execution in noninteractive mode.

The INCLUDE, AUTOCALL, COPY, OVERLAY and UNMAPCNT statements specify what object modules you want link-edited and whether you want the load module to execute in mapped or unmapped storage or as an overlay. The remaining control statements affect the initiation and operation of the link-edit.

$EDXLINK - Linkage Editor *(continued)*

The following are the $EDXLINK control statements used to perform a link-edit.

- *
- AUTOCALL
- COPY
- END
- INCLUDE
- LINK
- OVLAREA
- OVERLAY
- RESET
- UNMAPCNT
- VOLUME.

When you specify your control statements, the following order is recommended.

- All INCLUDE control statements for the resident portion of the program.
- An OVERLAY control statement followed by all INCLUDE control statements for that overlay. The OVERLAY control statement is terminated by another OVERLAY, an AUTOCALL, or a LINK control statement.
- The AUTOCALL control statement can appear anywhere prior to the LINK control statement.
- The LINK control statement initiates the link edit. It must follow all control statements specifying the modules to be linked together.

Notes:

1. Each control statement, with the exception of the OVLAREA statement, may be abbreviated down to a minimum of two characters. OVLAREA is abbreviated as OVL.
2. A control statement starts in column one, and you must separate any operands by one or more blanks.
3. You may use remarks on all control statements except for the AUTOCALL control statement. There must be at least one blank separating the control data and the remark.
4. You may not continue control statement data on a second line.

$EDXLINK - Linkage Editor *(continued)*

* comment Statement

Use the * comment statement to identify comments or remarks.

Syntax:

```
* THIS IS A COMMENT STATEMENT
```

AUTOCALL Statement

Use the AUTOCALL statement to identify autocall data sets. You can specify up to three autocall data sets. The linkage editor searches the autocall data sets in the order you specify on the AUTOCALL statement. You may not include remarks on the AUTOCALL statement.

Syntax:

```
AUTOCALL name,volume name,volume name,volume

Required:  none
Default:   no autocall processing
```

If you specify the AUTOCALL statement multiple times, the last one is the one $EDXLINK uses. You can correct an error in the AUTOCALL statement by reentering the AUTOCALL statement. You can turn off the autocall option by specifying the AUTOCALL statement without an autocall data set. For more information see "AUTOCALL Option" on page UT-294.

Examples:

```
AUTOCALL $PLIAUTO
AUTOCALL $AUTO,ASMLIB
AUTOCALL $PLIAUTO,EDX002 $AUTO,ASMLIB
AUTOCALL MYAUTO,MYVOL $PLIAUTO,EDX002 $AUTO,ASMLIB
AUTOCALL                  <= turns autocall option off
```

$EDXLINK - Linkage Editor *(continued)*

COPY Statement

Use the COPY statement to identify a *secondary-control-statement data set* from which to get additional control statements. The following control statements are permitted in a COPY data set:

- *
- INCLUDE
- OVLAREA
- OVERLAY
- VOLUME

The system flags all other control statements with warning messages and ignores them. See "Using $EDXLINK Control Statement Data Sets" on page UT-295 for information on using existing $EDXLINK data sets.

Syntax:

```
COPY name,volume

Required: name
Default:  volume defaults (see VOLUME)
```

Examples:

```
COPY NEXTOVLY
COPY MORECNTL,EDX003
```

$EDXLINK - Linkage Editor *(continued)*

END Statement

Use the END statement to identify the end of the control-statement data set. If the system encounters END in a *primary-control-statement data set* or if you enter it while in interactive mode, $EDXLINK ends. If the system encounters END in a *secondary-control-statement data set*, it ends processing of the COPY data set.

Syntax:

```
END

Required:  none
Default:   none
```

INCLUDE Statement

Use the INCLUDE statement to identify the object module(s) that $EDXLINK is to include in the generated output program. If you specify multiple object modules on the INCLUDE statement, you must include the volume name.

Syntax:

```
INCLUDE name1,name2,name3,name4,......,volume

Required:  name1
Default:   if a single data set name is entered, volume
           defaults to the IPL volume (see VOLUME)
```

Examples:

```
INCLUDE OBJECT1                          <= defaults to IPL volume
INCLUDE OBJECT2,EDX003                   <= one data set on EDX003
INCLUDE OBJECT3,OBJECT4,OBJECT5,EDX003   <= multiple data sets on
                                            volume EDX003
```

$EDXLINK - Linkage Editor *(continued)*

LINK Statement

Use the LINK statement to perform a link using the control statements previously processed. The name on the LINK statement is the name of the executable program to be generated. Upon completion of a link, $EDXLINK returns to process more link control statements.

Syntax:

```
LINK name,volume REPLACE END

 Required:  name
 Default:   volume defaults (see VOLUME)
```

The REPLACE option causes the linkage editor to replace any program with the same name as the one specified on the LINK statement. If you do not specify REPLACE (or R) and a program with the same name exists, you are prompted for REPLACE. If you answer YES to the prompt, the linkage editor replaces the program. If you answer NO to the prompt, the linkage editor prompts for a new program data set name. If you do not wish to generate the program, enter END and $EDXLINK does not generate a program but generates a link map. The END option ends $EDXLINK after completion of the link.

You must consider the following when you invoke $EDXLINK through $JOBUTIL:

1. When invoked through $JOBUTIL, $EDXLINK runs in noninteractive mode. To replace an existing program with a newly generated program, be sure that you coded the REPLACE option on your LINK statement.

2. To ensure that the $EDXLINK completion code generated as a result of the link edit is returned to $JOBUTIL, you must include an END statement in the last LINK statement.

Examples:

```
LINK PGMTEST
LINK SORT,EDX003
LINK PATCH,ASMLIB REPLACE
LINK PATCH2,ASMLIB REPLACE END
LINK GRAPH2,ASMLIB END
```

$EDXLINK - Linkage Editor *(continued)*

OVLAREA Statement

Use the OVLAREA statement to specify an overlay area used by overlay segments defined with the OVERLAY statement. You control the size and location of the overlay area by specifying the starting address (ENTRY in the root segment) and the ending address (ENTRY in the root segment). If you do not specify an overlay area, $EDXLINK automatically generates an overlay area large enough to contain the largest overlay segment specified in your program.

Syntax:

```
OVLAREA strtaddr endaddr

Required: strtaddr endaddr
Default:  none
```

Operands	*Description*
strtaddr	Starting address (ENTRY in root segment) of the overlay area.
endaddr	Ending address (ENTRY in root segment) of the overlay area.

$EDXLINK - Linkage Editor *(continued)*

OVERLAY Statement

Use the OVERLAY statement to identify the overlay segments. All the INCLUDE statements before the first OVERLAY statement make up the root or resident segment of the program. After the first OVERLAY statement, all INCLUDE statements until the next OVERLAY statement make up the first overlay segment, and so on until the system processes the LINK statement. If you enter an OVERLAY immediately following an OVERLAY statement, the second one overrides the first one.

Syntax:

```
OVERLAY REUSABLE

 Required: none
 Default:  not reusable
```

You can specify that the overlay segment is reusable by coding REUSABLE on the OVERLAY statement. Not reusable is the default. If you code REUSABLE and issue a CALL to the overlay segment, the system loads the overlay only if it is not already in storage. The default 'not reusable' causes the system to load the overlay segment each time you issue a CALL to this overlay segment.

Examples:

```
OVERLAY
OVERLAY REUSABLE
```

RESET Statement

Use the RESET statement to reset $EDXLINK. The system ignores all previous control statements. RESET is only valid in interactive mode. You receive a warning if it is found in a control statement data set.

Syntax:

```
RESET

 Required: none
 Default:  none
```

$EDXLINK - Linkage Editor *(continued)*

UNMAPCNT Statement

Use the UNMAPCNT statement to specify the number of overlay segments associated with a specific program that will reside in unmapped storage. This statement causes $EDXLINK to bring in an overlay manager to handle the overlay segments in unmapped storage.

If you do not specify the number of overlay segments or specify only one overlay segment, the system ignores the UNMAPCNT statement (no unmapped storage will be used) and the link proceeds.

If the link is a supervisor link, the system does not allow unmapped storage and $EDXLINK issues the following message:

```
*** WARNING - NO UNMAPPED STORAGE CAN BE USED FOR A SUPERVISOR
```

Syntax:

```
UNMAPCNT number

Required:  number
Default:   none
```

Operands	*Description*
number	The number of overlay segments to reside in unmapped storage.

Example:

```
UNMAPCNT 5
```

$EDXLINK - Linkage Editor *(continued)*

VOLUME Statement

Use the VOLUME statement to set the default volume for all $EDXLINK control statements. Initially the default is the IPL volume. If you don't specify a volume name on the VOLUME statement, the system resets the default to the IPL volume.

Syntax:

```
VOLUME volume

 Required: none
 Default:  IPL volume
```

Examples:

```
VOLUME EDX003
VOLUME OBJVOL
VOLUME                    <= reset to IPL volume
```

$EDXLINK Primary-Control-Statement Data Set

The following is an example of a multilink $EDXLINK *primary-control-statement data set.* The first link, Figure 16 on page UT-285, specifies that the executable program be linked without overlays. The second link, shown on the following page, specifies that the executable program be linked with overlays. It also references a *secondary-control-statement data set*, Figure 17 on page UT-286 .

This example is shown in two parts for illustration purposes. You normally create it as one complete data set.

$EDXLINK - Linkage Editor *(continued)*

```
* PLOT PROGRAM INCLUDES
*
INCLUDE PLOTXY,MYVOL
INCLUDE PLOTXX,MYVOL
INCLUDE PLOTYY,MYVOL
INCLUDE PLOTYX,MYVOL
*
* PERFORM AUTOCALL PROCESSING USING:
*
AUTOCALL MYAUTO,MYVOL $AUTO,ASMLIB
*
* PERFORM THE LINK
*
LINK PLOT,MYVOL REPLACE
* ROOT SEGMENT INCLUDES
*
INCLUDE SORT1,EDX003
INCLUDE SORT2,SORT3,SORT4,SORT5,OBJVOL
INCLUDE SAVEMOD,BACKUP
*
* OVERLAY SEGMENT #1 INCLUDES
*
OVERLAY
  INCLUDE SEARCH,OBJVOL
  INCLUDE SCRATCH,BACKUP
  INCLUDE READSUB,WRITESUB,EDX003
*
* OVERLAY SEGMENT #2 WILL BE REUSABLE
*
OVERLAY REUSABLE
  INCLUDE PATCH,OBJVOL
**************************************************************
* GET OVERLAYS #3 AND #4 FROM SECONDARY CONTROL STATEMENT   *
* DATA SET (see Figure 17 on page UT-286)                      *
**************************************************************
COPY PACKUNPK,MYVOL
* PERFORM AUTOCALL PROCESSING USING:
*
AUTOCALL $AUTO,ASMLIB
*
* PERFORM THE LINK AND TERMINATE $EDXLINK
*
LINK SORT,EDX003 REPLACE END
```

Figure 16. Multilink $EDXLINK primary-control-statement data set

$EDXLINK - Linkage Editor *(continued)*

$EDXLINK Secondary-Control-Statement Data Set

The following is an example of a $EDXLINK *secondary-control-statement data set*. This *secondary-control-statement data set* was referenced in the previous example as PACKUNPK in the COPY statement.

```
*
* THESE OVERLAYS ARE REQUIRED WHEN DATA
* COMPRESSION/DECOMPRESSION IS USED
*
OVERLAY
  INCLUDE PACKDATA,BIGBUFF,OBJVOL
OVERLAY
  INCLUDE UNPKDATA,BIGBUFF,OBJVOL
END
```

Figure 17. $EDXLINK Secondary Control Statement Data Set

Specifying Dynamic Storage

To increase program performance, you can change the dynamic storage used by $EDXLINK. $EDXLINK is shipped with an execution storage size of 22K. You can split this storage size (22K) between the program and dynamic storage. You can reduce the dynamic storage to a minimum of 8K (8192 bytes). If you specify any amount less than 8K, you will get unpredictable results.

The following examples show how to change an 8K dynamic allocation area to 10K temporarily. The first example shows changing dynamic storage using the $L command. The second example shows changing dynamic storage using the $JOBUTIL utility.

```
> $L $EDXLINK,,10240
```

OR

```
EXEC      STORAGE=10240
```

$EDXLINK - Linkage Editor *(continued)*

To change dynamic storage permanently, load $DISKUT2 and perform the following:

```
> $L $DISKUT2
LOADING $DISKUT2      49P,04:50:15, LP= 0000

USING VOLUME EDX002

COMMAND(?):  SS
PROGRAM NAME:  $EDXLINK
ENTER NEW STORAGE SIZE IN BYTES  10240

OLD STORAGE SIZE WAS         8192

NEW SIZE WILL BE          10240

OK TO CONTINUE?  Y

COMMAND(?):  EN

$DISKUT2 ENDED AT 04:50:35
```

Invoking $EDXLINK

You can invoke $EDXLINK with the $L operator command, the job stream processor ($JOBUTIL), or option 7 of the session manager.

Invoking $EDXLINK with the $L Operator Command

The following are examples of invoking $EDXLINK with the $L operator command. (For examples of using the session manager, see the *Event Driven Executive Language Programming Guide*.) After you load $EDXLINK, the system prompts you for execution parameters (PARM). There are two parameters that you may specify for use with $EDXLINK:

1. name,volume - specifies the *primary-control-statement data set*, and

2. printer - specifies where to direct the printed output.

The two parameters are positional. The *primary-control-statement data set* name must come first, optionally followed by the printer name. The printer name can be separated by one blank only from the name of the primary control statement data set name. If you do not code a printer name, it defaults to $SYSPRTR. If you enter END for the *primary-control-statement data set* name, $EDXLINK ends.

To run $EDXLINK interactively, enter an asterisk (*) for the *primary-control-statement data set* name.

$EDXLINK - Linkage Editor *(continued)*

Invoking with $L Noninteractively: The following is an example of invoking $EDXLINK noninteractively. The LINKWORK data set, used by $EDXLINK as a work file, must contain at least 256 records.

```
> $L $EDXLINK
 LINKWORK(NAME,VOLUME):  LINKWORK,EDX002
LOADING $EDXLINK    88P,00:00:33, LP=0000, PART=1

$EDXLINK - EDX LINKAGE EDITOR

PARM (?):  ?

  ENTER A LINK CONTROL DATA SET NAME,VOLUME OR AN
  ASTERISK (*) FOR INTERACTIVE MODE.  A PRINTER
  DEVICE NAME MAY OPTIONALLY BE ENTERED AFTER THE
  LINK CONTROL DATA SET NAME,VOLUME.

PARM (?):  LINK1,EDX003 MPRINTER

$EDXLINK CONTROL STATEMENT PROCESSING STARTED
$EDXLINK EXECUTION STARTED
CASE     ,EDX003 STORED
PROGRAM DATA SET SIZE =   39
COMPLETION CODE =  -1


$EDXLINK ENDED AT 00:01:12
```

Invoking with $L Interactively: The following is an example of a multilink interactive session. The storage parameter is specified to take advantage of $EDXLINK'S use of dynamic storage. This example is broken into three parts for illustrative purposes. You normally create it as one data set.

Part 1 displays all the $EDXLINK control statements, links one module, contains no overlays, and runs to a successful completion (-1).

$EDXLINK - Linkage Editor *(continued)*

```
> $L $EDXLINK,,20000
 LINKWORK(NAME,VOLUME):  LINKWORK,EDX002
LOADING $EDXLINK    120P,04:00:59, LP=0000, PART=1

$EDXLINK - EDX LINKAGE EDITOR


PARM (?):   * $SYSLOG

$EDXLINK INTERACTIVE MODE
  DEFAULT VOLUME = EDX002

STMT (?):  ?

$EDXLINK CONTROL STATEMENTS
  AUTOCALL - SPECIFY AUTOCALL DATA SET(S)
  COPY     - COPY THE SPECIFIED CONTROL STATEMENTS FROM DSNAME,VOLUME
  END      - TERMINATE $EDXLINK
  INCLUDE  - INCLUDE THE FOLLOWING MODULE(S)
  LINK     - GENERATE AN EXECUTABLE PROGRAM
  OVERLAY  - SPECIFY THE BEGINNING OF AN OVERLAY SEGMENT
  OVLAREA  - SPECIFY LOCATION OF OVERLAY AREA
  RESET    - RESET $EDXLINK
  VOLUME   - CHANGE DEFAULT VOLUME
  UNMAPCNT - SPECIFY NUMBER OF UNMAPPED STORAGE OVERLAYS

STMT (?):   INCLUDE TESTSUB4,EDX003

STMT (?):   LINK CASE1,EDX003 REPLACE

$EDXLINK EXECUTION STARTED
CASE1    ,EDX003 STORED
PROGRAM DATA SET SIZE =    12
COMPLETION CODE =  -1
```

Figure 18. $EDXLINK multilink interactive interface

Part 2 of this example is the same as part 1 except that it links multiple data sets. They are all listed on the same INCLUDE control statement because they reside on the same volume.

```
$EDXLINK INTERACTIVE MODE
  DEFAULT VOLUME = EDX002

STMT (?):   INCLUDE OBJ12,OBJ13,OBJ14,OBJ15,EDX003

STMT (?):   LINK PGM1,EDX003 REPLACE

$EDXLINK EXECUTION STARTED
PGM1     ,EDX003 STORED
PROGRAM DATA SET SIZE =    15
COMPLETION CODE =  -1
```

Part 3 of this example contains multiple overlay segments. The root section is named TESTROOT, and the three overlays are TESTSUB1,TESTSUB2, and TESTSUB3. TESTSUB3 is reusable. The

$EDXLINK - Linkage Editor *(continued)*

AUTOCALL data set is used in this example, and an END statement is specified on the LINK statement to end the $EDXLINK session.

```
$EDXLINK INTERACTIVE MODE
  DEFAULT VOLUME = EDX002

STMT (?):  INCLUDE TESTROOT,EDX003

STMT (?):  OVERLAY

STMT (?):  INCLUDE TESTSUB1,EDX003

STMT (?):  OVERLAY

STMT (?):  INCLUDE TESTSUB2,EDX003

STMT (?):  OVERLAY REUSABLE

STMT (?):  INCLUDE TESTSUB3,EDX003

STMT (?):  AUTOCALL $AUTO,ASMLIB

STMT (?):  LINK TEST,EDX003 REPLACE END

$EDXLINK EXECUTION STARTED
TEST    ,EDX003 STORED
PROGRAM DATA SET SIZE =   26
COMPLETION CODE =   -1

$EDXLINK ENDED AT 04:05:35
```

Invoking $EDXLINK with the Job Stream Processor

The following are examples of invoking $EDXLINK with the job stream processor ($JOBUTIL). You can specify two parameters on the $JOBUTIL parameter statement:

1. name, volume - specifies the *primary-control-statement data set*, and

2. printer - specifies where to direct the printed output.

The two parameters are positional. The *primary-control-statement data set* must come first, optionally followed by the printer name. The printer name can be separated by one blank only from the primary-control-statement data set name. If you do not code a printer name, it defaults to the $SYSPRTR.

To run $EDXLINK interactively from $JOBUTIL, enter an asterisk (*) for the *primary-control-statement data set* name.

$EDXLINK - Linkage Editor *(continued)*

Invoking with $JOBUTIL Noninteractively: In this example, a *primary-control-statement data set* called LINK1 on EDX003 is specified and the printed output is defaulted to $SYSPRTR. The storage parameter is also specified to take advantage of $EDXLINK's use of dynamic storage.

```
JOB        LINK2
PROGRAM    $EDXLINK
DS         LINKWORK,EDX002
PARM       LINK1,EDX003
EXEC       STORAGE=20000
EOJ
```

Invoking with $JOBUTIL Interactively: In this example, $EDXLINK is to be run interactively. The printed output is directed to MPRINTER.

```
JOB        LINK1
PROGRAM    $EDXLINK
DS         LINKWORK,EDX002
PARM       * MPRINTER
EXEC
EOJ
```

Operator Termination of $EDXLINK

To terminate $EDXLINK before a logical end of job, use the attention interrupts CA and RE.

The CA attention interrupt is active at all times and returns control to the PARM (?): prompt. You then have the option of ending $EDXLINK or starting a new session.

The RESET (RE) control statement is only active in $EDXLINK interactive mode and returns control to the STMT (?): prompt. You then have the option of ending $EDXLINK or starting a new link.

Note: If you are in interactive mode and processing control statements, the STMT (?): prompt may appear before the cancel or reset take effect. Press the enter key again and the cancel or reset will proceed normally.

$EDXLINK - Linkage Editor *(continued)*

$EDXLINK Output

The link map is a listing that $EDXLINK generates. This listing corresponds with the example Figure 18 on page UT-289. The example is broken into three parts for illustrative purposes. You will see it as one listing when it prints.

Part 1 shows all data sets included in the link-edit, and the message output generated by the execution of the link-edit.

```
$EDXLINK EXECUTION CONTROL STATEMENTS          04:01:13
  FROM INTERACTIVE MODE                        03/25/81

INCLUDE TESTROOT,EDX003
OVERLAY
INCLUDE TESTSUB1,EDX003
OVERLAY
INCLUDE TESTSUB2,EDX003
OVERLAY REUSABLE
INCLUDE TESTSUB3,EDX003
AUTOCALL $AUTO,ASMLIB
LINK TEST,EDX003 REPLACE END

$EDXLINK EXECUTION STARTED
*** WARNING - UNRESOLVED WEAK EXTERNAL REFERENCES
  SVC       SUPEXIT   SETBUSY
TEST    ,EDX003 STORED
PROGRAM DATA SET SIZE =   26
COMPLETION CODE =  -1
```

Figure 19. $EDXLINK map

Part 2 shows the storage layout of the root section and the overlay segments. It shows their entry points and sizes. The overlay area ($OVLAREA) is determined by the largest overlay segment rounded up to the next page boundary. This is the section from which you get the overlay segment number.

$EDXLINK - Linkage Editor *(continued)*

```
TEST    ,EDX003 STORED

RESIDENT SEGMENT:
              LABEL     ADDR  LNTH  LABEL     ADDR  LABEL  ADDR
    SECTION =           0000  027A
      ENTRY = BUFFER    0178        FLAG      0278

  AUTOCALL MODULES(S)
              LABEL     ADDR  LNTH  LABEL     ADDR  LABEL  ADDR
    SECTION = $OVLMGR   027A  00B6
    SECTION = $OVLCT    0330  0136
      ENTRY = $OVLDSCB  0330        $OVLSIZE  038C   SUB1  0444
      ENTRY = SUB3      03FC        SUB2      0420
    SECTION = $OVLAREA  0466  0200
                              ----
              * TOTAL   ===>  0666

OVERLAY SEGMENT:   1
              LABEL     ADDR  LNTH  LABEL     ADDR  LABEL  ADDR
    SECTION =           0466  0102
      ENTRY = SUB1      0468
                              ----
                TOTAL   ===>  0102

OVERLAY SEGMENT:   2
              LABEL     ADDR  LNTH  LABEL     ADDR  LABEL  ADDR
    SECTION =           0466  01D3
      ENTRY = SUB2      0468        SUB2A     0512
                              ----
                TOTAL   ===>  01D3

OVERLAY SEGMENT:   3
              LABEL     ADDR  LNTH  LABEL     ADDR  LABEL  ADDR
    SECTION =           0466  01F2
      ENTRY = SUB3      0468
                              ----
                TOTAL   ===>  01F2
```

Part 3 shows the GLOBAL section that is mapped to $SYSCOM when the program is loaded.

```
GLOBAL SECTION(S):
    SECTION = COMMAREA  0000  0028
                              ----
                TOTAL   ===>  0028

END OF MAP
```

$EDXLINK - Linkage Editor *(continued)*

AUTOCALL Option

The autocall option enables you to include modules that are not included explicitly via the INCLUDE statement. To invoke the AUTOCALL option, enter an AUTOCALL before the LINK statement specifying one to three autocall data sets.

Autocall Data Set

The autocall data set contains a list of object module names and volumes along with their entry points.

Autocall Data Set Record Format

The format of the autocall data set record is shown below. Each record must contain an object module "name,volume" beginning in column one and followed by at least one entry point. You must insert at least one blank between the object module "name,volume" and the entry point, and at least one blank between each entry point. If you have more entry points than can fit in one record, specify an additional record beginning with the object module "name,volume" followed by the remaining entry points. Finally, you must end the autocall data set with **END in either the object module name field or an entry point field.

The following are examples of the format of the autocall data set record.

```
SORT,EDX003  INPUT   OUTPUT RETCODE
PATCH,OBJVOL ADDRESS LENGTH TYPE
**END
```

System Autocall Data Set

The system autocall data set $AUTO is distributed with the Event Driven Executive Program Preparation Facility and installed on the volume ASMLIB. The data set contains the name of the modules and entry points that you can autocall as a result of including the following functions in your program:

- graphics formatting instructions
- data formatting instructions
- square root function
- screen formatting subroutines.

$EDXLINK - Linkage Editor *(continued)*

Autocall Processing

If you specify any autocall data sets and unresolved external references (EXTRN'S) remain after normal INCLUDE statement processing, the linkage editor searches the autocall data sets in the sequence that you specified. If external references match entry points in an autocall data set, the indicated object module is included with the resident segment. The linkage editor terminates autocall processing when no unresolved external references (EXTRN'S) remain after INCLUDE statement processing or no matches are found in any of the autocall data sets.

Unresolved External References

The system lists all unresolved external references after the $EDXLINK processing messages. Any unresolved EXTRNS cause $EDXLINK to return a completion code of 4.

Unresolved weak external references (WXTRN'S) do not invoke autocall processing. Unresolved WXTRN'S have no effect upon the $EDXLINK completion code.

Using $EDXLINK Control Statement Data Sets

To use existing $EDXLINK control statement data sets without changing them, do the following in interactive mode:

- Enter the COPY statement followed by the $EDXLINK control statement data set name and volume. A warning message appears in reference to the OUTPUT statement from the $EDXLINK control statement data set, causing the system to return a completion code of 4.
- If the OUTPUT statement had the AUTO= parameter coded, enter the AUTOCALL statement followed by the autocall data set name and volume that appeared on the OUTPUT statement.
- Enter the LINK statement with the executable program name and volume.

Example:

```
$EDXLINK INTERACTIVE MODE
  DEFAULT VOLUME = EDX002

STMT (?):  COPY PGMLINK,SRCVOL
OUTPUT   PGM1,EDX003 MAP AUTO=$AUTO,ASMLIB
*** WARNING - INVALID CONTROL STATEMENT - STATEMENT IGNORED.

STMT (?):  AUTOCALL $AUTO,ASMLIB

STMT (?):  LINK PGM,EDX003 REPLACE END

$EDXLINK EXECUTION STARTED
PGM     ,EDX003 STORED
PROGRAM DATA SET SIZE =    15
COMPLETION CODE =   4
```

$FONT - Process Character/Images Tables

$FONT creates or modifies the character image tables for the 4978 and 4980 display stations and the 4974 matrix printer. The image stores in these devices contain bit patterns that are interpreted by the hardware to produce the display or printing of characters. Each character bit pattern corresponds to an EBCDIC code, which is defined by a dot matrix coded into eight bytes of data for the 4974 and 4978 and 16 bytes of data for the 4980. You can change Character bit patterns to alter the appearance of existing characters in a device's image store or to create entirely new characters.

For the 4978 and 4980, you can modify the system-supplied image store data sets $4978IS0 and $4980IS0 by using $FONT. The system automatically loads these data sets to every 4978 and 4980 display station supported by the supervisor when you IPL your system. Any changes you make to the system-supplied data sets are reflected in all the 4978 or 4980 display stations attached to your Series/1. If you create a new image store and save it in a data set different than the system-supplied data set, you can load the image store to the display station of your choice.

You can change a 4978, 4980, or 4974 image table using either a 4978, 4980, or 4979 display station.

Data Set Requirements

To create a new image store, you must allocate an image store data set before using $FONT. The system uses this data set to save the newly-created image store. Use $DISKUT1 to allocate the data set.

Depending upon the size of the image store (device-dependent) you intend to create, we recommend the following data set sizes according to device type:

4974	768 bytes (3 records)
4978	2048 bytes (8 records)
4980	4096 bytes (16 records)

Invoking $FONT

You invoke $FONT with the $L operator command or option 4.5 of the session manager.

```
> $L $FONT
 LOADING $FONT     66P,17:01:11, LP= 0000, PART= 3
```

After you load $FONT, it prompts you for the name of an existing image store.

```
 DATA SET (NAME,VOLUME):
```

$FONT - Process Character/Images Tables *(continued)*

After you enter the data set name and volume, $FONT issues the following message:

```
DATA SET FORMATS:

    1) 4980
    2) 4978
    3) 4974

SELECT OPTION:
```

Enter the appropriate option depending upon the type of image store you intend to create or modify. The device type must match the format of the image-store to be read into the storage work area reserved by $FONT. For example, if you intend to modify a 4980 image store, select option 1.

Once you have selected the appropriate option, $FONT reads the contents of the image store data set into the storage work area.

$FONT Commands

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

```
COMMAND (?):  ?

DISP -- DISPLAY IMAGES
READ -- READ IMAGE DATASET
SAVE -- WRITE IMAGE DATASET
EDIT -- ENTER EDIT MODE
PUT  -- PUT IMAGE STORE INTO DEVICE
GET  -- GET IMAGE STORE FROM DEVICE
END  -- END $FONT

COMMAND (?):
```

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

You can enter the commands in abbreviated format; for example, you can enter R instead of READ. Each command and its explanation is presented in alphabetical order on the following pages. Each description shows the abbreviation and full word for each command.

DISP (D)— Display Images

Use the DISP command to display the characters generated for each EBCDIC code by the associated bit patterns in the image store. There are 256 EBCDIC codes. Not all devices support this number of characters. The utility displays only the characters that it supports, leaving the rest blank. (For descriptions of the characters supported by the 4974 matrix printer and the 4978/4980 display stations, see the *4974 Printer Description*, the *4978 General Information*, and the *4980 General Information* manuals.)

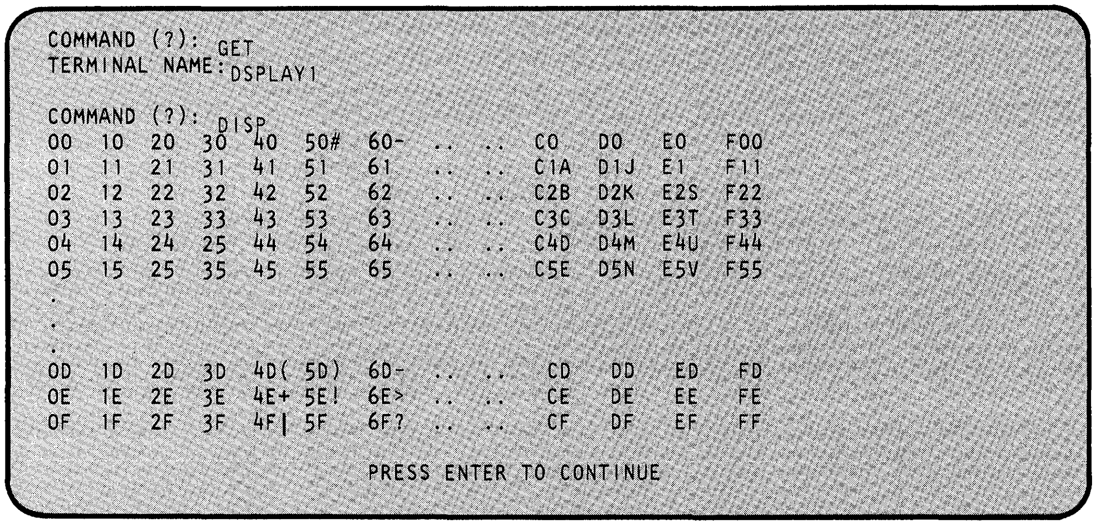
$FONT - Process Character/Images Tables *(continued)*

Notes:

1. You can restore the image buffer (wire image table) of the 4974 to the standard 64-character set by using the RE command of $TERMUT2.

2. You can restore the image buffers for the 4978 and 4980 using the LI of $TERMUT2.

Example: This screen display is in compressed format and does not show the entire image table as it would appear. Note the use of the GET command prior to DISP.



```
COMMAND (?): GET
TERMINAL NAME:DSPLAY1

COMMAND (?): DISP
00  10  20  30  40  50#  60-  ..  ..  C0   D0   E0   F00
01  11  21  31  41  51   61   ..  ..  C1A  D1J  E1   F11
02  12  22  32  42  52   62   ..  ..  C2B  D2K  E2S  F22
03  13  23  33  43  53   63   ..  ..  C3C  D3L  E3T  F33
04  14  24  25  44  54   64   ..  ..  C4D  D4M  E4U  F44
05  15  25  35  45  55   65   ..  ..  C5E  D5N  E5V  F55
.
.
.
0D  1D  2D  3D  4D( 5D)  6D-  ..  ..   CD   DD   ED   FD
0E  1E  2E  3E  4E+ 5E!  6E>  ..  ..   CE   DE   EE   FE
0F  1F  2F  3F  4F| 5F   6F?  ..  ..   CF   DF   EF   FF

                        PRESS ENTER TO CONTINUE
```

The system displays the EBCDIC characters to the right of their associated hexadecimal codes.

To end display mode, press the enter key; $FONT issues the COMMAND (?): prompt again. Then you can enter the EDIT command to modify or create a character.

Notes:

1. Whenever you use a 4979 to modify an image store, the DISP command displays the character set of the 4979.

2. If you display a character set formatted for one type of terminal on a different type of terminal (for example, if you display a 4980 character set on a 4978), the message CHARACTER APPROXIMATIONS appears at the bottom of the display. This means that the characters displayed are only representations for the actual characters defined for the terminal.

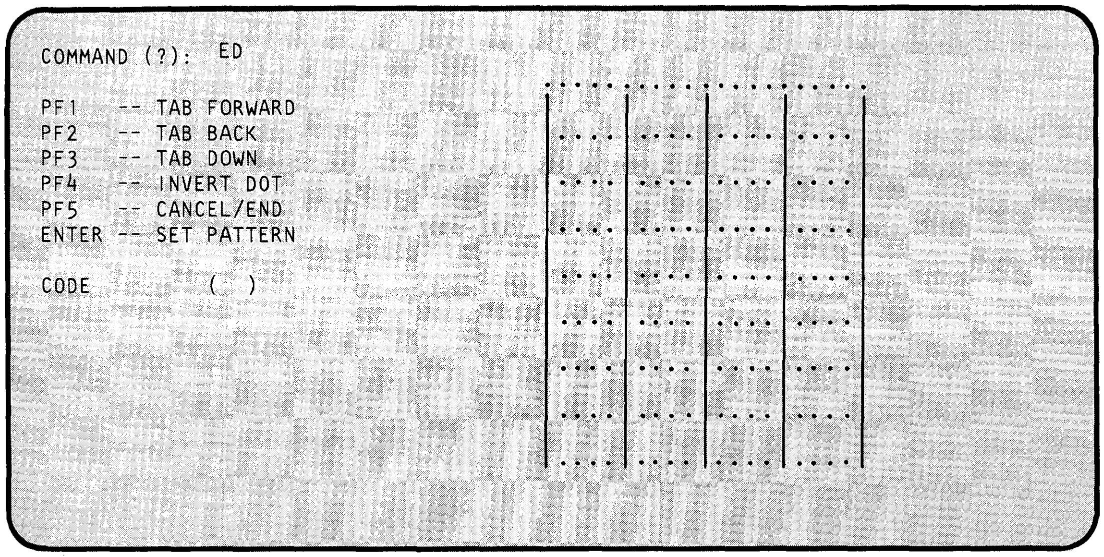
$FONT - Process Character/Images Tables *(continued)*

EDIT (ED) — Enter Edit Mode

Use the EDIT command to create a new image table or to modify an existing image table. Depending upon the device type, $FONT displays a dot matrix pattern (a 4 x 8 grid for the 4974 and 4978 or an 8 x 14 grid for the 4980). You can specify the character you want to appear on the grid for modifications. The image on the grid shows the changes you make as you modify or create a character.

After you finish creating a new image table or modifying an existing one, you need to save the image table or load it into a device. Use the SAVE command to save it in a specified data set or the PUT command to load the table into a specified device.

Example: Edit a 4980 image control data set.



```
COMMAND (?): ED

PF1   -- TAB FORWARD
PF2   -- TAB BACK
PF3   -- TAB DOWN
PF4   -- INVERT DOT
PF5   -- CANCEL/END
ENTER -- SET PATTERN

CODE         ( )
```

For an explanation of changing or creating characters, see "Create/Modify a Character Image" on page UT-306.

$FONT - Process Character/Images Tables *(continued)*

END (EN) — End the $FONT Utility

Use the END command to end the $FONT utility.

```
COMMAND (?): EN
```

If you created or modified an image store and did not save the contents of the storage work area, $FONT prompts you as follows:

```
SAVE TABLE?
```

If you respond **Y,** $FONT prompts you for the name of the data set in which to save the contents of the storage work area.

```
DATA SET (NAME,VOLUME):
```

This gives you another opportunity to save the contents of storage work area.

If you respond **N,** $FONT ends. The contents of the work area are no longer available.

$FONT - Process Character/Images Tables *(continued)*

GET (G)— Get Image Store from Device

Use the GET command to get the contents of an image store from a 4978, 4980, or 4974 terminal and place them into the storage work area reserved by $FONT. After you get the image store into the work area, you can display it with the DISP command or modify it using the EDIT command.

```
COMMAND (?):  GET devname
```

Enter the name of the device (specified on the TERMINAL statement) whose image store you want read into the work area.

If you do not specify a device name on the same line as the GET command, $FONT prompts you as follows:

```
ENTER TERMINAL NAME (CR OR * = THIS ONE):
```

If the system cannot find the requested terminal, $FONT issues the following message:

```
NOT ONLINE, ANOTHER NAME?
```

If you respond **Y,** $FONT prompts you for another terminal name. If you respond **N,** $FONT issues the COMMAND (?): prompt.

If you enter the name of a terminal that is not a 4978, 4980, or 4974, $FONT issues the following messages:

```
UNSUPPORTED DEVICE

COMMAND (?):
```

Examples:

```
COMMAND (?):  GET
ENTER TERMINAL NAME:  DSPLAY1



COMMAND (?): GET DSPLAY1
```

$FONT - Process Character/Images Tables *(continued)*

PUT (P) — Put Image Store into Device

Use the PUT command to load a device with the contents of the work area.

Notes:

1. The image table will revert to its original state when you initialize the system again or if a power failure occurs.

2. You can also load a specified device with an altered or new image table by using the LI command (load an image store) of the $TERMUT2 utility.

```
COMMAND (?): PUT device
```

Enter the name of the device (specified on the TERMINAL statement) you want loaded with the new or modified image table.

If you do not specify a device name on the same line as the PUT command, $FONT prompts you as follows:

```
ENTER TERMINAL NAME (CR OR * = THIS ONE):
```

If you attempt to load a 4974 printer with an image store greater than 96 characters (for example, a 4978 image store of 98 characters), $FONT issues the following message:

```
THE FIRST 96 NONBLANK CHARACTERS OF THE
IMAGE STORE WILL BE PUT TO THE 4974.

CONTINUE? (Y/N)
```

If you respond **Y**, the system loads into the 4974 only the first 96 nonblank characters it encounters. If you respond **N,** $FONT issues the COMMAND (?): prompt.

$FONT - Process Character/Images Tables *(continued)*

If you attempt to load a 4974 with a 4980 image data set, $FONT sends the characters corresponding to a fixed set of 96 EBCDIC codes. These codes are listed in the following message:

```
THE EBCDIC HEX CODES OF THE 4980
CHARACTERS ABOUT TO BE PUT TO THE 4974 ARE:

4A ... 4F
50, 5A ... 5E
60, 61, 6A ... 6F
7A ... 7F
81 ... 89, 8F
91 ... 99
A1 ... A9
B0, BA
C0 ... C9
D0 ... D9
E0, E2 ... E9
F0 ... F9

CONTINUE? (Y/N)
```

If you respond **Y**, the system loads the EBCDIC characters associated with the hexadecimal codes into the 4974. If you respond **N,** $FONT issues the COMMAND (?): prompt again.

Examples:

```
COMMAND (?):  PUT
TERMINAL NAME:  DSPLAY1


COMMAND (?):  PUT DSPLAY1
```

$FONT - Process Character/Images Tables *(continued)*

READ (R) — Read Data Set

Use the READ command to read the contents of an image store data set into the storage work area reserved by $FONT. $FONT prompts you for the data set format.

```
DATA SET FORMATS:

      1) 4980
      2) 4978
      3) 4974

SELECT OPTION:
```

The device type must match the format of the image-store data set. After the system reads the image store into the storage work area, you can display it by using the DISP command or modify it by using the EDIT command.

Example: Read an image store into the storage work area.

```
COMMAND (?): READ MYDATA,MYVOL

DATA SET FORMATS:

      1) 4980
      2) 4978
      3) 4974

SELECT OPTION:
```

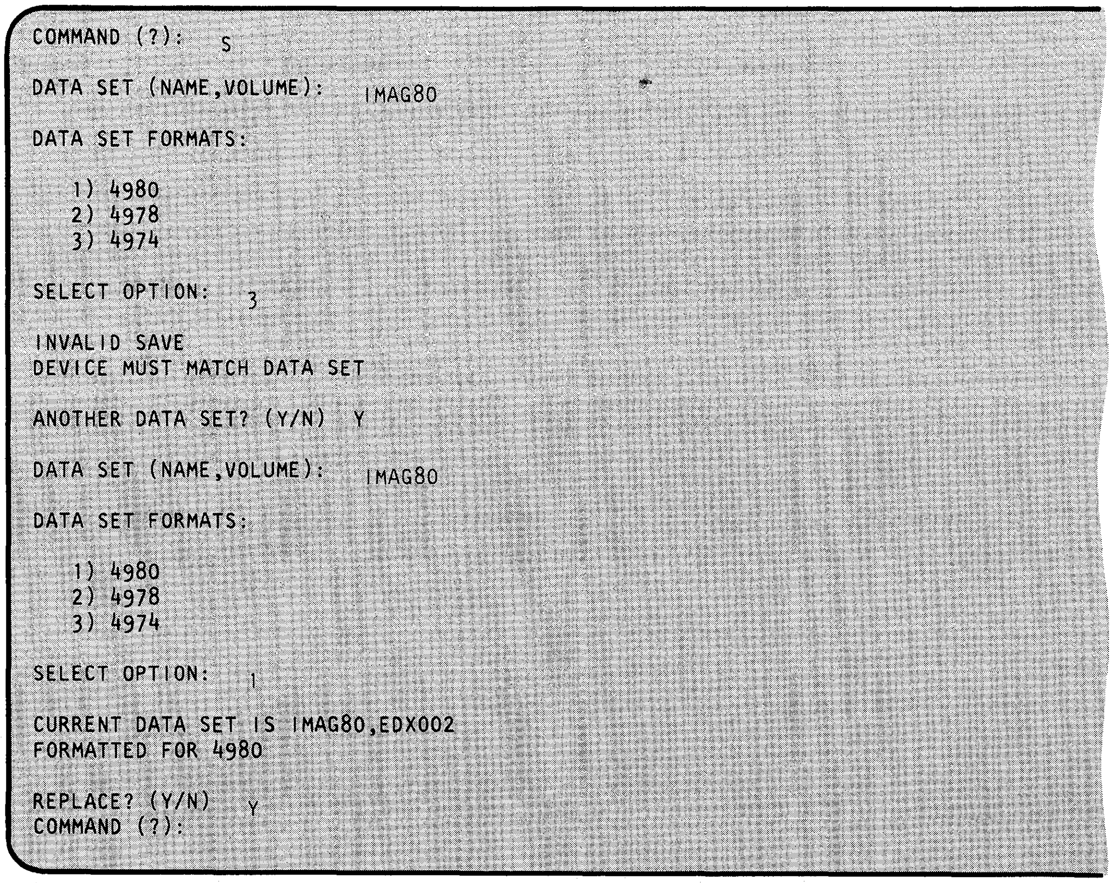
If you don't specify the volume name, it defaults to the volume from which $FONT was loaded.

$FONT - Process Character/Images Tables *(continued)*

SAVE (S) — Write Image Data Set

Use the SAVE command to save the contents of the storage work area reserved by $FONT to a specified data set. The format of the data set must match the format of the image store currently in the storage work area. You can use the SAVE command to initialize an empty data set or to replace the contents of a data set that has been used before. If you save the contents in an existing data set, you no longer have access to the previous contents.

Example: In the following example, the first attempt to save an image store IMAG80 (a 4980 image store) in a data set formatted for a 4974 is invalid. $FONT prompts for another data set. The second save attempt is successful because the work data set format matches the current image store type (in this case, a 4980 image store).



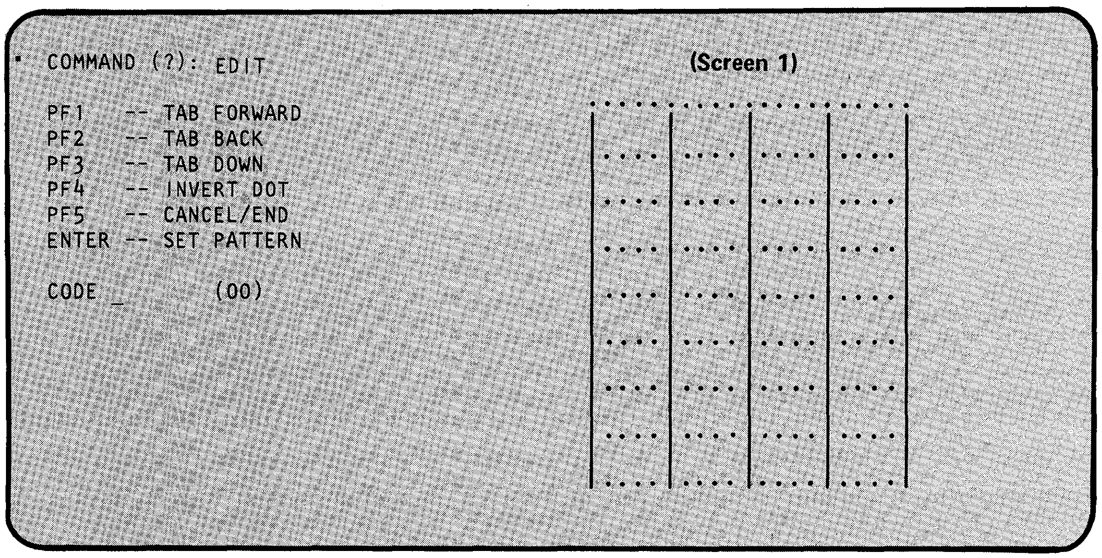
```
COMMAND (?):  S

DATA SET (NAME,VOLUME):   IMAG80

DATA SET FORMATS:

   1) 4980
   2) 4978
   3) 4974

SELECT OPTION:   3

INVALID SAVE
DEVICE MUST MATCH DATA SET

ANOTHER DATA SET? (Y/N)  Y

DATA SET (NAME,VOLUME):   IMAG80

DATA SET FORMATS:

   1) 4980
   2) 4978
   3) 4974

SELECT OPTION:   1

CURRENT DATA SET IS IMAG80,EDX002
FORMATTED FOR 4980

REPLACE? (Y/N)   Y
COMMAND (?):
```

$FONT - Process Character/Images Tables *(continued)*

Create/Modify a Character Image

Use the EDIT command to make changes to characters or to create new characters. In edit, you can choose a specific character to be displayed on a grid and can make changes to that character. The grid shows all changes. The grid remains blank until you specify the EBCDIC character and/or hexadecimal code.

In this example, a 4978 or 4974 image store is to be modified.



```
COMMAND (?): EDIT                                   (Screen 1)

PF1   -- TAB FORWARD
PF2   -- TAB BACK
PF3   -- TAB DOWN
PF4   -- INVERT DOT
PF5   -- CANCEL/END
ENTER -- SET PATTERN

CODE _        (00)
```

Note the position of the cursor: it is located just to the right of the CODE prompt. Enter the character you wish to modify next to the CODE prompt. Enter the character's corresponding hexidecimal code in the parentheses to the right of the cursor.

Function of the PF Keys

Use the PF keys listed on the screen to move the cursor and make modifications to the character on the grid. The PF keys and their functions are as follows:

PF1 Moves the cursor forward (left-to-right and top-to-bottom) across the grid.

PF2 Moves the cursor backwards within the grid (right-to-left and bottom-to-top).

PF3 Moves the cursor from its present line down to the next line.

PF4 Inverts the dot pattern.

PF5 Returns you to command mode or cancels the current edit session. Press PF5 once to cancel the edit session. Press PF5 a second time to return to command mode.

$FONT - Process Character/Images Tables *(continued)*

Function of the Enter Key

The enter key has an important use when you're editing. Press the enter key when you are finished making changes to the character on the grid. $FONT issues the following prompt:

```
REPLACE    x  (xx)?
```

where x represents the EBCDIC character being modified and xx is the hexadecimal equivalent of the character. Enter **Y** to make the changed character part of the image store. Enter **N** if you want to associate the modified character with a different EBCDIC character or hexadecimal code. $FONT issues the following message:

```
SET TO NEW CODE?
```

If you respond **Y,** $FONT prompts you for the new code as follows:

```
NEW CODE:  _  (00)
```

Enter the EBCDIC character and/or hexadecimal code with which you want the modified character associated. For example, if you want to associate a modified A (C1) with the B character, enter B (C2).

If you respond **N,** $FONT assumes you do not wish to save the changed character and positions the cursor at the first line, leftmost corner of the grid. You can make additional changes to the character currently in the grid or press PF5 to move the cursor down to the CODE __ (00) prompt. Then enter the EBCDIC character and hexadecimal code of another character you wish to modify.

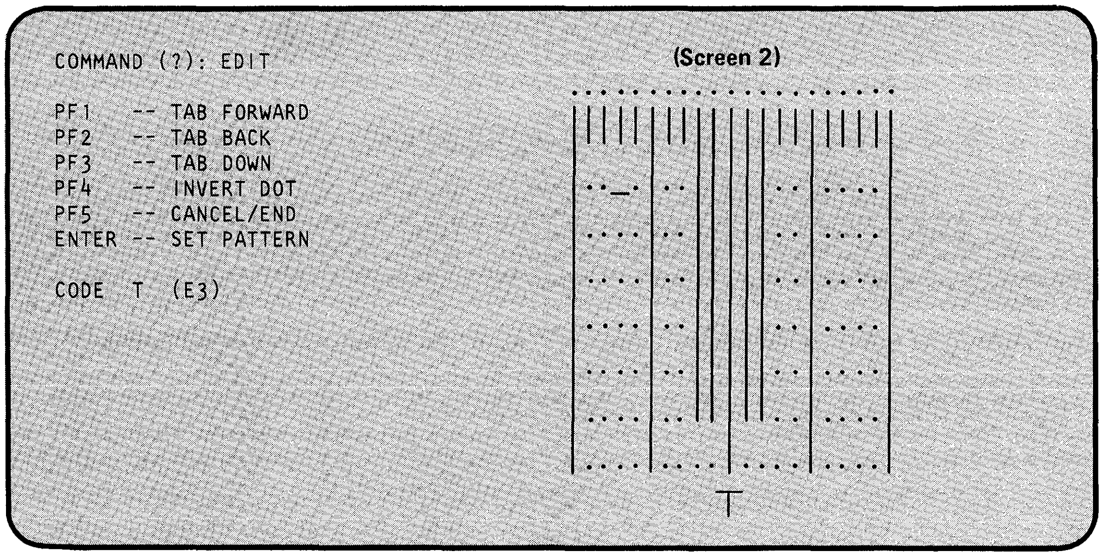
$FONT - Process Character/Images Tables *(continued)*

Example of Modifying a Character

To specify a character to appear on the grid for modification:

1. Enter the desired character. You can also move the cursor to the right and enter the corresponding hexadecimal code, if known, between the parentheses. This example shows the letter T and its corresponding hexadecimal code.

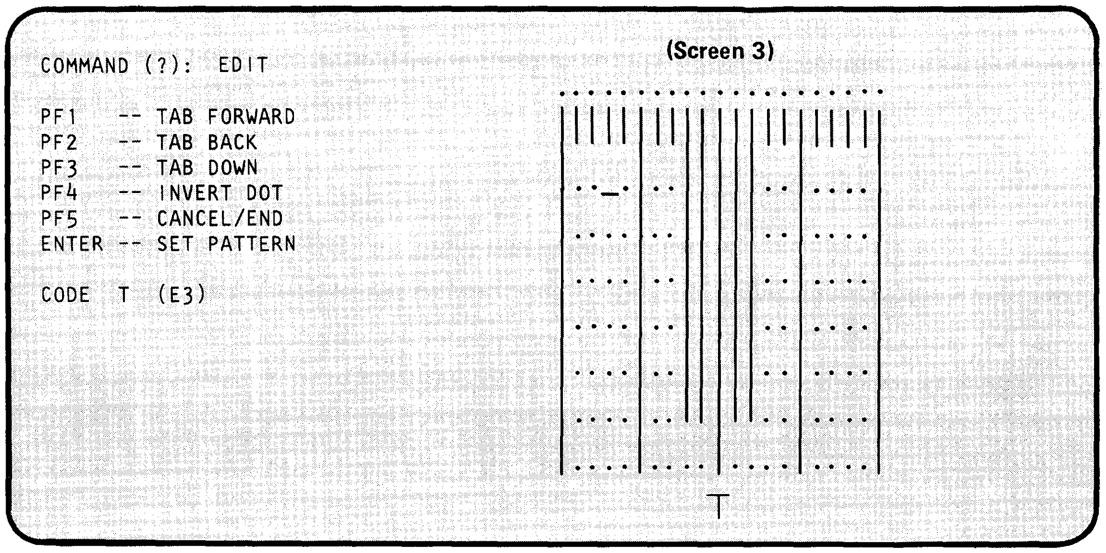
2. $FONT then fetches the bit pattern for the specified EBCDIC character or hexadecimal code and displays the corresponding character image in the grid. Screen 2 shows the letter, its hexadecimal code, and the resulting image of the T on the grid.



```
COMMAND (?): EDIT

PF1    -- TAB FORWARD
PF2    -- TAB BACK
PF3    -- TAB DOWN
PF4    -- INVERT DOT
PF5    -- CANCEL/END
ENTER  -- SET PATTERN

CODE  T  (E3)
```

$FONT - Process Character/Images Tables *(continued)*

3. After the image appears on the grid, the cursor moves to the top left-hand square of the grid. The hexadecimal code for the character on the grid remains displayed in the parentheses to the right of the CODE prompt, as shown in Screen 3. ($FONT also displays the character in its actual size below the grid. The appearance of this character changes to reflect the changes to the character on the grid.)



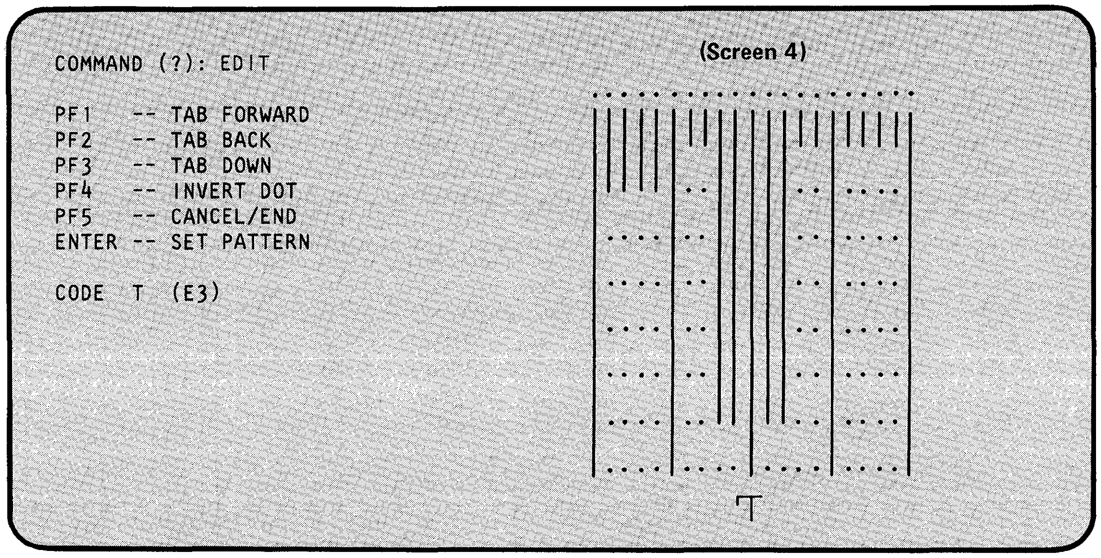
(Screen 3)

```
COMMAND (?):  EDIT

PF1    -- TAB FORWARD
PF2    -- TAB BACK
PF3    -- TAB DOWN
PF4    -- INVERT DOT
PF5    -- CANCEL/END
ENTER  -- SET PATTERN

CODE  T  (E3)
```

4. With the cursor positioned as shown in Screen 3, use the PF keys to modify the letter T. In this case, you are extending the top left and right sections of the crossbar.

$FONT - Process Character/Images Tables *(continued)*

5. Press the PF4 key; you have inverted the dot pattern in the first square and extended the left end of the crossbar downward. The image on the grid changes as shown in Screen 4:

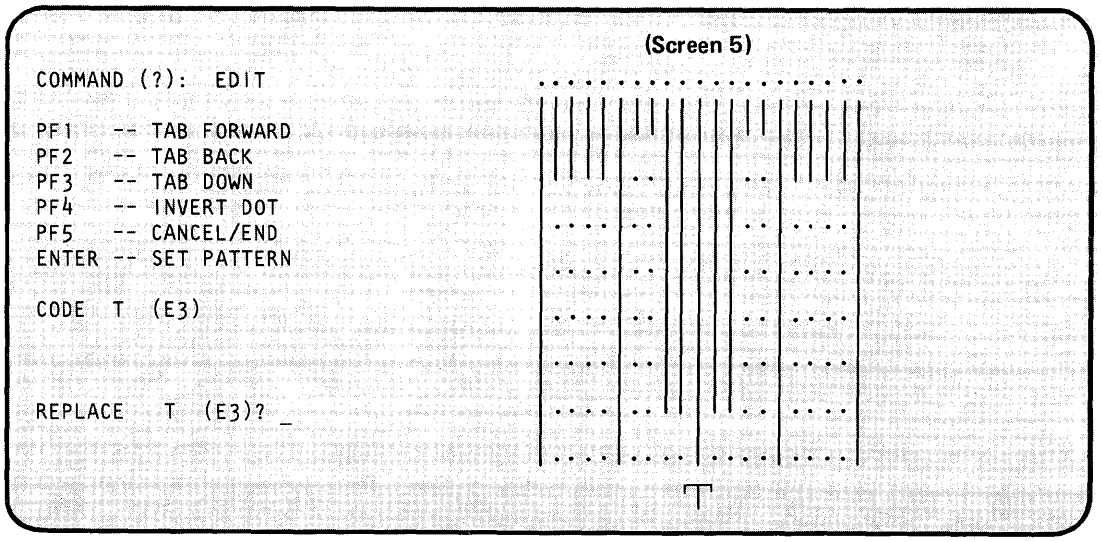


```
COMMAND (?): EDIT

PF1   -- TAB FORWARD
PF2   -- TAB BACK
PF3   -- TAB DOWN
PF4   -- INVERT DOT
PF5   -- CANCEL/END
ENTER -- SET PATTERN

CODE  T  (E3)
```

6. Press the PF1 key to move the cursor across the grid into the top right square. Press the PF4 key again to extend the right crossbar downward, as was done on the left side.

$FONT - Process Character/Images Tables *(continued)*

7. Once changes to the character are complete, press the enter key. This changes the CODE prompt to REPLACE and positions the cursor just to the right of the REPLACE prompt, as shown in Screen 5.



(Screen 5)

```
COMMAND (?):  EDIT

PF1    -- TAB FORWARD
PF2    -- TAB BACK
PF3    -- TAB DOWN
PF4    -- INVERT DOT
PF5    -- CANCEL/END
ENTER  -- SET PATTERN

CODE  T  (E3)


REPLACE   T  (E3)? _
```

Now, you can either "set" the modified character as it appears on the grid into the image table, "cancel" all changes just made to the character, or "associate" the character with a key other than the one specified.

1. To "set" the modified character into the image table, respond **Y** to the REPLACE prompt to replace the original T in the image table.

2. To "cancel" the changes made to the character, respond **N** to the REPLACE prompt and the SET TO NEW CODE prompt.

3. To "associate" the character with a character other than the one specified, respond **N** to the REPLACE prompt and then **Y** to the SET TO NEW CODE prompt. $FONT then prompts you for the EBCDIC character and hexadecimal code you wish to associate with the modified character.

After any of the above actions, the REPLACE prompt changes back to the CODE prompt and clears the grid, ending that editing session. The blank screen reappears (Screen 1) and you can specify another character for modification.

To end edit mode, press PF5 and reenter command mode. If you want to be able to load the modified image table to a 4978, 4974, or 4980 at a future time, you must store the image table in a disk or diskette data set. Use the SAVE command to store the new or modified image in the data set specified.

$FSEDIT - Full Screen Editor

$FSEDIT is a full-screen text editing utility that helps you develop and modify source data. It operates the terminal as a static screen device and, therefore, must be run from a terminal with static screen capability (for example, 4978, 4979, 4980, or 3101 display terminals).

With $FSEDIT you can:

- Edit or browse a data set using a full screen.
- Scroll information forward and backward.
- Use PF (program function) keys for frequently-used functions.
- Insert a mask for prefilling inserted lines.
- Merge data from other data sets.
- Print a data set.
- Display a directory list for a specific volume.
- Communicate with a System/370 in conjunction with the Host Communication Facility IUP installed on the host System/370.

Note: To use $FSEDIT, the following modules must reside on the the same volume as $FSEDIT: $FSMEN, $FSEDB, $FSUT1 to $FSUT4, $FSIMA, and $FSIM0 to $FSIM8.

Invoking $FSEDIT

Invoke $FSEDIT with the $L operator command or option 1 of the session manager. If you are loading $FSEDIT from an application program or by using a $JOBUTIL procedure, you must pass $FSEDIT a 1-word parameter of zeroes.

$FSEDIT requires a preallocated work data set. The system automatically allocates this work data set if you invoke $FSEDIT with the session manager. If you use the $L operator command to invoke $FSEDIT, the system prompts you for the name of the work file in the following manner:

```
> $L $FSEDIT
 WORKFILE (NAME,VOLUME):
```

See "Work Data Set" on page UT-315 for a description of the work data set.

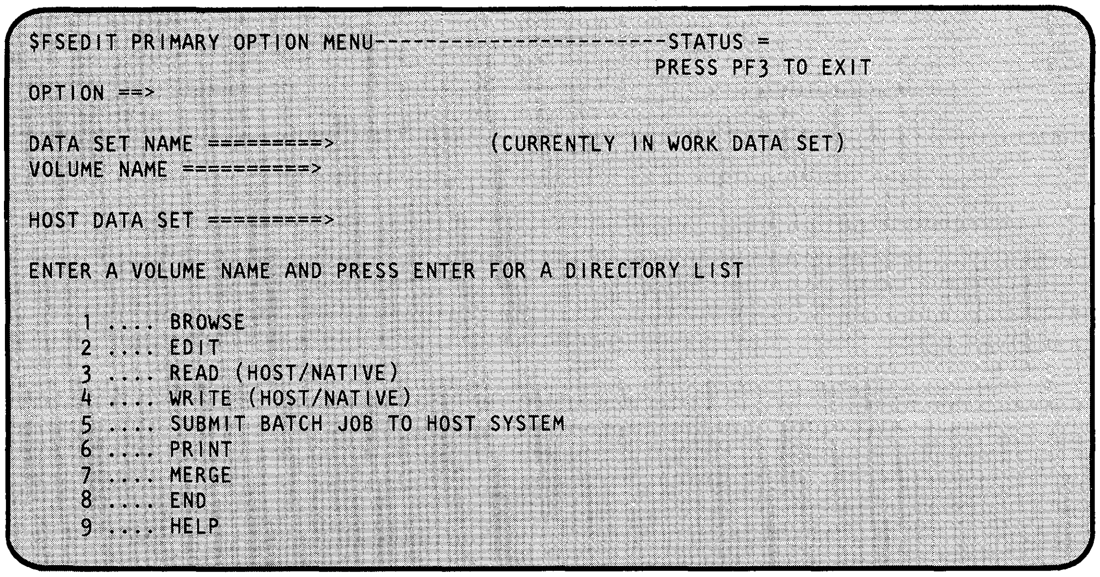
$FSEDIT - Full Screen Editor *(continued)*

After you enter the name of the work data set and press the enter key, $FSEDIT issues the following message if this is the first time you have used the work data set.

```
DS1 HAS NOT PREVIOUSLY BEEN USED AS A WORK DATA SET
IS IT OK TO USE IT NOW?
```

Respond Y. If you respond N, $FSEDIT ends.

Once you load $FSEDIT, it displays the following primary option menu. You then have the option of selecting one of the primary options or displaying a directory list for a specific volume.



```
$FSEDIT PRIMARY OPTION MENU-----------------------STATUS =
                                                 PRESS PF3 TO EXIT
OPTION ==>

DATA SET NAME =========>            (CURRENTLY IN WORK DATA SET)
VOLUME NAME ==========>

HOST DATA SET =========>

ENTER A VOLUME NAME AND PRESS ENTER FOR A DIRECTORY LIST

     1 .... BROWSE
     2 .... EDIT
     3 .... READ (HOST/NATIVE)
     4 .... WRITE (HOST/NATIVE)
     5 .... SUBMIT BATCH JOB TO HOST SYSTEM
     6 .... PRINT
     7 .... MERGE
     8 .... END
     9 .... HELP
```

The primary option menu has four input fields. They are the:

- OPTION field
- DATA SET NAME field
- VOLUME NAME field
- HOST DATA SET field

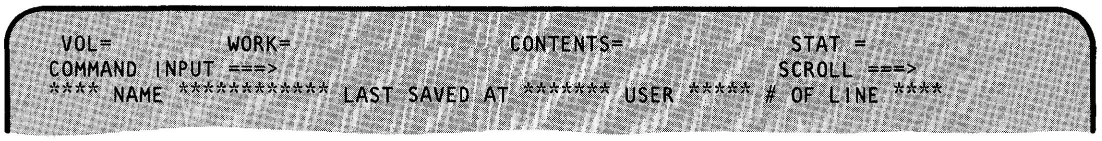
Select Primary Options

To select one of the $FSEDIT primary options, enter an option number in the OPTION field. For example, if you enter a data set name and volume in the DATA SET NAME and VOLUME NAME fields and the number 3 in the OPTION field, $FSEDIT reads the contents of the specified data set into the work data set. All subsequent $FSEDIT commands act upon the work data set.

$FSEDIT - Full Screen Editor *(continued)*

Select Directory List

You can also look at the list of directory entries (data-type data sets) on a specific volume. To do this, leave the OPTION field blank, specify the name of a volume in the VOLUME NAME field, and press the enter key. $FSEDIT displays a directory data set list of the specified volume.



```
 VOL=         WORK=                    CONTENTS=            STAT =
COMMAND INPUT ===>                                        SCROLL ===>
**** NAME ************ LAST SAVED AT ******* USER ***** # OF LINE ****
```

The first line of the screen contains the following information:

VOL =	The volume for which the system displays the directory data set list.
WORK =	The name of the work data set.
CONTENTS =	The name of the source data set whose contents were copied into the work data set.
STAT =	The status of the contents of the work data set per the last action performed:

UNCHANGED	The contents of the work data set have not been modified.
INIT	The work data set has been initialized.
MODIFIED	The contents of the work data set have been modified.
SAVED	The contents of the work data set have been saved.

After $FSEDIT displays the directory list, you can use the directory data set list commands or you can browse or edit any of the data sets. See "Directory Data Set List Commands" on page UT-319 for a description of each command.

To browse or edit a specific data set in the list, position the cursor next to the data set name in the directory list, enter B (browse) or E (edit), and press the enter key. $FSEDIT reads the contents of the data set into the work data set. You can now edit or browse the information in the work data set. (See "Option 1 — BROWSE" on page UT-327and"Option 2 — EDIT" on page UT-327 for a description of the BROWSE and EDIT commands.)

To obtain a hard copy of the directory data set list, use the $DISKUT1 utility (LA or LISTP command).

$FSEDIT - Full Screen Editor *(continued)*

Work Data Set

$FSEDIT requires a preallocated work data set for use as a text edit work area. If you invoked $FSEDIT through the session manager, the session manager automatically allocates the work data set. Text data (source statements) within this work data set are in a special text editor format, identical to that used by the $EDIT1N text editor. See "Data Set Requirements" on page UT-235. You can edit data within a work data set with either $EDIT1N or $FSEDIT.

Notes:

1. $FSEDIT uses source data sets of 80-character lines that are line numbered in positions 73-80 for host or Series/1 data sets. Positions 1-72 contain source data, and positions 73-80 contain sequence numbers assigned by $FSEDIT for host or Series/1 data sets. The system pads these source statements to 128 bytes, and $FSEDIT stores them in source statement format (two 128-byte statements in each 256-byte record).

   The format of a $FSEDIT readable EDX record for source data sets is:

Bytes	Words	Contents
0-71	1-36	Source Line
72-79	37-40	Line Number
80-127	41-64	Zeros (reserved for future use)
128-199	65-100	Source Line
200-207	101-104	Line Number
208-255	105-128	Zeros (reserved for future use)

2. The work data set should not be used by any other program.

3. The $FSEDIT utility uses a data set as a work area. During an edit session, the system saves the information you entered on the screen in a buffer area. Once this buffer area fills up, the system saves the contents in the work data set. If you IPL your system before you save the contents of the buffer in the work data set, the contents are lost. To avoid losing information in the buffer area, use option 4 (WRITE) before IPLing your system. If you write the work data set to a tape, you no longer have access to previously existing data sets on that tape.

When you end an editing session, save the contents of the work data set in a source data set on disk, diskette, or tape. To save the work data set contents, use the WRITE command while in Edit mode or return to the primary menu and select option 4 (WRITE). If the target data set does not exist on the volume specified, $FSEDIT creates it automatically. The system performs automatic translation from text editor format to source statement format.

$FSEDIT - Full Screen Editor *(continued)*

Scrolling

The information in the work data set and the directory data set list usually exceeds the size of the display screen. Scrolling allows you to page up or down through the information. Two PF keys are used for this purpose, one for each direction. A scroll amount is displayed at the end of the second line of the edit, browse, or directory list screen showing the number of lines scrolled with each use of a scroll key. The scroll amounts are:

PAGE or P	Specifies scrolling one page (22 lines).
HALF or H	Specifies scrolling a half page (11 lines).
MAX or M	Specifies scrolling to the top or bottom of the data set.
n	Specifies number of scrolling lines.

In browse mode, the scroll amount is initialized to PAGE; in edit mode, it is initialized to HALF.

You can change the scroll amount by moving the cursor to the scroll field and entering an amount over what is currently displayed. To change the scroll amount, type over the first character with a P, H, or M to change the scroll amount to a page, half page, or maximum, respectively. To specify a number of lines to scroll, enter the number of lines you want.

When you make a change, the new value remains in effect for the remainder of the session unless you change it again. The value for MAX is an exception; following a single MAX scroll, the scroll amount defaults back to the value for browse mode (PAGE) or edit mode (HALF).

$FSEDIT - Full Screen Editor *(continued)*

Program Function Keys

The ATTN and six program function keys are used to request commonly used or special $FSEDIT operations as follows:

PF1	Redisplays the screen image. All changes are ignored and the original data is displayed as last entered before you pressed the PF key.
PF2	SCROLL UP — scrolls towards the beginning of the data set by the amount shown in scroll amount field.
PF3	SCROLL DOWN or END UTILITY — scrolls toward the end of the data set by the amount shown in the scroll amount field in edit mode or ends the $FSEDIT utility.
PF4	REPEAT FIND — repeats the action of the previous FIND primary command.
PF5	REPEAT CHANGE — repeats the action of the previous CHANGE primary command (applies only to edit mode).
PF6	PRINT SCREEN — prints the screen image on the system printer ($SYSPRTR) or the hard-copy device defined with the HDCOPY= operand on the TERMINAL statement. You can redirect the output to another printer using the PS (print screen) command. If the screen is printed, the message SCREEN PRINTED appears in the upper right hand column of the screen. If the printer is busy, the message PRINTER BUSY appears in the upper right hand column of the screen.
ATTN CA	Cancel the print option. Pressing the attention key and typing CA stops the list option of $FSEDIT and returns to the primary menu.

Notes:

1. The PF2 - PF4 and PF6 keys are active only during browse and edit modes. Using PF1 and PF5 is only meaningful during edit mode.

2. If you are using the starter system as your operating system and you have a 4979 display station at address 04, the program function keys operate as follows:

PF1	Redisplays the screen image.
PF2	Repeat find.

$FSEDIT - Full Screen Editor *(continued)*

PF3	Scroll up.
PF4	Repeat change.
PF5	Scroll down.
PF6	Print screen.

If you are using a 3101 terminal, the following additional program function keys are used:

PF7	(Insert Mode only). This key performs the same function as the SEND key. Due to hardware restrictions, you must use this key as a line terminator if you are performing multiple inserts.
PF8	This key performs the same function as the 'ATTN' key on the 4978, 4979, and 4980 display terminals.

3101 Display Terminal Switch Settings

If you are using a 3101 display terminal to edit or create programs, you can avoid operational problems with the 3101 terminal by setting the 3101 switches correctly. Refer to the *Installation and System Generation Guide* for detailed information on switch settings. Valid switch settings for your installation are found on the underside of the switch cover plate.

If you require that the DUAL/MONO switches be set to DUAL, you can force command and program directives for the utilities or application programs to upper case by:

- Pressing the shift key or
- Using the CAPS command for the $FSEDIT edit session.

$FSEDIT - Full Screen Editor *(continued)*

You should consider the following if you select DUAL case mode.

- In an environment where there is a mix of terminal types, some terminals may not display lowercase characters.
- Some printer types do not print lowercase characters.

Directory Data Set List Commands

$FSEDIT can display a list of the directory entries (data-type data sets that contain EBCDIC characters) on a specific volume. You can browse the list, sort the list by data set name or last date the data set was changed, locate a specific data set, or edit or browse the work data set or a specific entry on the list.

To perform these functions, the following commands are available:

B (BOTTOM)	Scroll to the bottom of directory list.
BR (BROWSE)	Browse contents of the work data set.
CV	Change volume.
ED (EDIT)	Edit the contents of the work data set.
E (END)	Return to the previous menu.
L (LOCATE)	Locate a specific data set name.
M (MENU)	Return to the previous menu.
S (SORT)	Sort directory data set list.
T (TOP)	Scroll to the top of directory list.

In addition to these commands, there are two line commands that you can use when displaying the directory data set list: B (BROWSE) and E (EDIT).

With the exception of the B (BROWSE) and E (EDIT) line commands, you can activate each of these commands by entering its name next to the COMMAND INPUT prompt. See "Directory Line Commands" on page UT-325 for an explanation of using the B and E line commands.

You can enter most of these commands in full or abbreviated form. For example, you can enter BR instead of BROWSE. The abbreviated and full word command for each command, if applicable, are shown on the following pages.

$FSEDIT - Full Screen Editor *(continued)*

B (BOTTOM) - Display Bottom of Directory List

Use the B (BOTTOM) command to display the end (bottom) of the directory list. This is equivalent to scrolling to the bottom of the list.

Syntax:

```
BOTTOM
B

Required: None
Defaults: None
```

BR (BROWSE) - Browse Contents of the Work Data Set

Use the BR (BROWSE) command to examine the contents of the work data set. This command is equivalent to primary option 1 (BROWSE). See "Option 1 — BROWSE" on page UT-327 for a description of the browse function.

Syntax:

```
BROWSE
BR

Required: None
Defaults: None
```

$FSEDIT - Full Screen Editor *(continued)*

CV — Change Volume

Use the CV command to change the volume you are accessing currently. When you enter the CV command and the name of a volume, $FSEDIT displays the directory data set list of the specified volume.

Syntax:

```
CV volname

Required:  volname
Defaults:  None
```

Operands	***Description***
volname	The name of a volume (1-6 characters)

$FSEDIT - Full Screen Editor *(continued)*

ED (EDIT) — Edit the Contents of the Work Data Set

Use the ED (EDIT) command to edit the contents of the work data set. This command is equivalent to primary option 2 (EDIT). See "Option 2 — EDIT" on page UT-327 for a description of the function.

Syntax:

```
EDIT
ED

Required:  None
Defaults:  None
```

E (END) — Return to Primary Option Menu

Use the E (END) command to end the current mode and return to the primary option menu.

Syntax:

```
END
E

Required:  None
Default:  None
```

$FSEDIT - Full Screen Editor *(continued)*

L (LOCATE) — Locate a Specific Data Set Name

Use the L (LOCATE) command to locate a specific data set name.

You can also enter a generic name, that is, a prefix instead of an entire name. $FSEDIT searches through the list and locates the first data set starting with the specified prefix.

Syntax:

```
LOCATE dsname
L     dsname

Required: dsname
Defaults: None
```

Operands	***Description***
dsname	Name of the data set you want to locate.

M (MENU) — Return to Primary Option Menu

Use the M (MENU) command to end the current mode and return to the primary option menu.

Syntax:

```
MENU
M

Required: None
Default:  None
```

$FSEDIT - Full Screen Editor *(continued)*

S (SORT) — Sort Directory Data Set List

Use the S (SORT) command to sort the display of the directory data set list. You can sort the list by date (last date the data set was updated) or by data set name (the default).

Note: If a volume has more than 250 members, you can only sort the directory data set list by data set name.

Syntax:

```
SORT option
S    option

Required: option
Defaults: None
```

Operands	***Description***	
option	There are three sort options:	
	DATE or D	Sort the list by the date the data set was last updated.
	NAME or N	Sort the list by data set name (the default).
	RESET or R	Reset the current value of the sort option to NO SORT.

T (TOP) — Display Top of Directory List

Use the TOP command to return to the beginning (top) of the directory data set list.

Syntax:

```
TOP
T

Required: None
Defaults: None
```

$FSEDIT - Full Screen Editor *(continued)*

Directory Line Commands

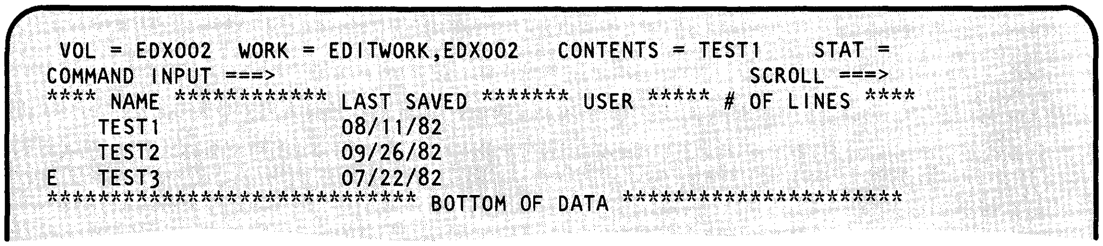
Use the following line commands to browse or edit a data set on the directory data set list.

B (BROWSE) Read the selected data set into the work data set and enter Browse mode.

E (EDIT) Read the selected data set into the work data set and enter Edit mode.

To select a specific data set, place the cursor next to the data set name in the directory list, enter E or B, and press the enter key. $FSEDIT reads the contents of the data set into the work data set. Now you can edit or browse the information in the work data set. See "Option 1 — BROWSE" on page UT-327 and "Option 2 — EDIT" on page UT-327 for a description of these commands.

Example: Select a data set from the directory list to edit.



```
 VOL = EDX002  WORK = EDITWORK,EDX002   CONTENTS = TEST1    STAT =
COMMAND INPUT ===>                                      SCROLL ===>
**** NAME ************ LAST SAVED ******* USER ***** # OF LINES ****
    TEST1              08/11/82
    TEST2              09/26/82
E   TEST3              07/22/82
***************************** BOTTOM OF DATA **********************
```

$FSEDIT - Full Screen Editor *(continued)*

Primary Options and Commands

Under $FSEDIT, the following primary options are displayed on the primary menu:

Option 1 - Browse	Browse the contents of the work data set.
Option 2 - Edit	Edit the contents of the work data set.
Option 3 - Read	Obtain a copy of a host/native data set and store it in the work data set.
Option 4 - Write	Transfer the contents of the work data set to a host/native data set.
Option 5 - Submit	Submit a job to the host.
Option 6 - Print	Print the contents of the work data set on the device designated as $SYSPRTR.
Option 7 - Merge	Merge all or part of a source data set with the contents of the work data set.
Option 8 - End	End the $FSEDIT utility.
Option 9 - Help	Display tutorial text for using the $FSEDIT utility.

A description of each option follows.

$FSEDIT - Full Screen Editor *(continued)*

Option 1 — BROWSE

Use option 1 (BROWSE) to enter browse mode. In BROWSE mode, you can look at source data in the work data set, but you cannot change it.

You can look at all parts of the work data set by using the PF2 and PF3 program function keys. These keys enable you to scroll forward or backward through the data set.

In addition, you can use two primary commands to locate specific information within the data set.

FIND	Searches for a designated text string.
LOCATE	Searches for a designated line number.

These primary commands are discussed under "Primary Commands" on page UT-334.

During browsing, the current number of lines in your data set and the maximum number of lines the work data set can hold are displayed in parentheses on the top line of the display, following the data set name and the volume identification.

To end browsing, enter the primary command END or MENU in the COMMAND INPUT field and return to the primary option menu.

Option 2 — EDIT

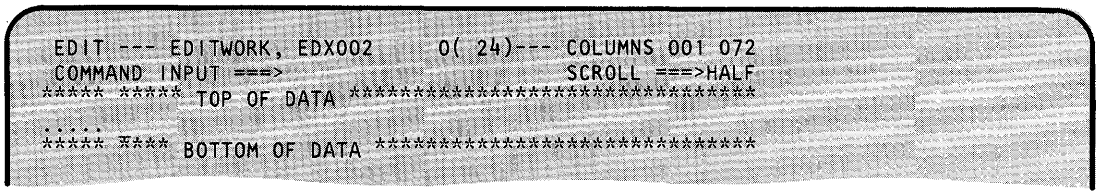
In EDIT mode, you can modify an existing source data set or create a new one. To do this, you use:

- Program function keys for two-way scrolling as well as repeat, change, and find;
- Primary commands (CAPS, CANCEL, COBOL, CHANGE, CLEAR, END, FIND, LOCATE, MENU, PS, RENUM, RESET, TABS, and WRITE);
- Line edit commands to manipulate whole lines or blocks of lines. (See "Edit Line Commands" on page UT-344 for an explanation of these commands.)

$FSEDIT - Full Screen Editor *(continued)*

Creating a Source Data Set

To create a new source data set, enter EDIT mode (option 2) with an empty data set (the work data set you specified when you invoked $FSEDIT) or use the CL (CLEAR) command to clear the current work data set while in edit mode. Because the work data set is empty, the editor assumes you desire insertion (creation) of lines and activates the INSERT function. The following is an example of the initial display when you are editing an empty data set.



```
 EDIT --- EDITWORK, EDX002     0( 24)--- COLUMNS 001 072
 COMMAND INPUT ===>                      SCROLL ===>HALF
***** ***** TOP OF DATA ********************************
.....
***** **** BOTTOM OF DATA ******************************
```

The top line of the screen, from left to right, displays utility mode (EDIT), the name and volume of the work data set (EDITWORK,EDX002), the number of source statements in the work data set, and the total number of statements the data set will hold (in parentheses). The COLUMNS 001 072 indicates that the source data can only be entered in columns 1 through 72.

The cursor is positioned at character position 1 of the insert line. After you enter information on this line, press the enter key. The utility then numbers the entered line and sets up for the next insert line.

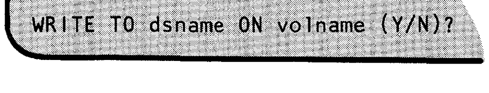
As you continue in this manner, a new insert line is readied each time the preceding line is entered (by pressing the enter key). You can end the insert (creation) operation by pressing the enter key without entering anything on the new insert line.

> **Note:** $FSEDIT does not distinguish between insert mode and edit mode during editing operations, and you can change data on the screen at any time.

There are two ways to save the source statements just created:

1. In edit mode, enter the command WRITE on the COMMAND INPUT line to save the contents of the work data set in the original (source) data set. If you want to save the work data set in a new data set, enter WRITE data set,volume.

2. Exit edit mode by entering an M on the COMMAND INPUT line and return to the primary option menu. Enter the number 4 (WRITE) on the OPTION line to save the contents of the work data set in the original (source) data set. If you want to save the work data set in a new data set, enter the data set and volume name in the DATA SET NAME and VOLUME NAME fields. $FSEDIT prompts you as follows:



```
WRITE TO dsname ON volname (Y/N)?
```

If you respond Y, the contents of the work data set are saved in the data set specified in the DATA SET NAME and VOLUME NAME fields. If the data set does not exist on the volume

$FSEDIT - Full Screen Editor *(continued)*

specified, $FSEDIT creates it automatically. If you respond N, $FSEDIT resets the menu and does not write the contents of the work data set to the specified data set.

Modifying an Existing Data Set

There are two ways to enter edit mode and modify an existing data set. If you use the primary option menu, the data set must be read first into the work data set using option 3 (READ). If you use the directory data set list, move the cursor down to the desired data set name, type 'E' next to it, and press the enter key. $FSEDIT reads the data set into the work data set.

Locate and change information by scrolling the data set by pressing the PF keys. The PF keys and definitions are described under "Program Function Keys" on page UT-317.

To modify data on the screen, move the cursor to the desired location and enter the new information. You can change several lines before pressing the enter key. You can delete, insert, duplicate, or rearrange a single line or a block of lines with the edit commands. These are discussed under "Edit Line Commands" on page UT-344.

For general editing purposes, you can use primary edit commands to find and change designated character strings and to change the line numbering sequence. These commands are discussed under "Primary Commands" on page UT-334.

After you finish modifying the contents of the work data set, you need to save it in the existing source data set or in another data set. You can perform the save from EDIT mode or from the primary option menu.

There are two way to save statements just modified:

1. In edit mode, enter the command WRITE on the COMMAND INPUT line to save the contents of the work data set in the original (source) data set. If you want to save the work data set in a new data set, enter WRITE data set,volume.

2. Exit edit mode by entering an M on the COMMAND INPUT line and return to primary option menu. Enter the number 4 (WRITE) on the OPTION line to save the contents of the work data set in the original (source) data set. If you want to save the work data set in a new data set, enter the data set and volume name in the DATA SET NAME and VOLUME NAME fields. $FSEDIT prompts you as follows:

```
WRITE TO dsname ON volname (Y/N)?
```

   If you respond Y, the system saves the contents of the work data set in the data set specified in the DATA SET NAME and VOLUME NAME fields. If the data set does not exist on the volume specified, $FSEDIT creates it automatically. If you respond N, $FSEDIT resets the menu and does not write the contents of the work data set to the specified data set.

You can end the editing by entering the END or MENU command in the COMMAND INPUT field. END returns you to the previous menu; MENU returns you to the primary option menu.

$FSEDIT - Full Screen Editor *(continued)*

Editing Upper-Case and Lower-Case Character Data

When you use option 3 (READ), the system checks the entire data set for lowercase characters. This is done because lowercase data cannot be edited on a 4979, a 4978 or 4980 with an uppercase-only control store, or a 3101 with switches set for upper case. For information on how to change the 4978 or 4980 to a lowercase store see "$TERMUT2 - Change Image/Control Store" on page UT-557. For information on 3101 switch settings, see *IBM 3101 Display Terminal Description,*, GA18-2033.

If the system detects lowercase characters and you want to edit the data set using EDIT command, the following message is issued:

```
WARNING, DATA SET ISN'T ALL UPPER-CASE CHARACTERS
DO YOU WANT TO CONTINUE?
```

Entering an N causes a return to the primary option menu or directory data set list.

Entering a Y causes the edit session to continue. If you decide to continue editing and are using an uppercase terminal, you lose only the lowercase data. To prevent the loss of data, you can issue CAPS ON ALL or CANCEL as the first primary command.

If you issue the CAPS ON ALL command, all data in the work file becomes upper case and no line commands or data changes are processed. The screen is reshown in upper case.

Note: Setting CAPS ON ALL does not place change flags on the lines that are converted to upper case.

If you use the CANCEL command, the edit session is cancelled, and you are returned to the primary option menu. If you use the END or MENU command, the screen is processed and all lowercase characters are lost in the work data set.

If the data set you read in is all uppercase data and you edit it adding lowercase data, the warning message is not issued. However, it is issued if the data set is written to source and then read in again.

If you do a READ command and then an EDIT, and the system detects lowercase data, the edit session begins in CAPS OFF mode. If the system finds no lowercase data, then the session begins in CAPS ON mode.

Notes:

1. Brackets [,], plus/minus ±, and bullet • EBCDIC representations on the 4978 and 4980 are treated as lowercase characters with no uppercase equivalents. The 4978, 4979, and 4980 convert any unrecognizable character into a blank.

$FSEDIT - Full Screen Editor *(continued)*

2. On the 3101, brackets [,], and not-signs are also treated as lower case and, when converted to upper case, are displayed as colons. The 3101 converts any unrecognizable character into a colon.

3. The conversion from lowercase to uppercase characters takes place when you enter the primary command CAPS ON ALL in edit mode, or you alter a line containing one of these characters while in edit mode with CAPS ON. You cannot display these characters on a 4978, 4979 or a 4980 with an uppercase-only control store or on a 3101 with switches set for upper case.

4. The 'not-sign' hexadecimal representation on a 4978/4979 is equivalent to the 'caret' on a 3101.

5. The 'double bar' on the 4978, 4979, and 4980 is equivalent to the 'logical OR' on a 3101.

Option 3 — READ

Option 3 (READ) retrieves a data set from either a host system or a data set on disk, diskette, or tape on the native Series/1 system and stores it in your work data set. The primary option menu remains on the display and the area below it is used to display how many lines were read from the source data set, the date and time of the last update to the data set, and your user identification if you invoked $FSEDIT through the session manager. The data set name entered must be fully qualified and must contain fixed-length, 80-byte records.

Line numbers for native data sets must be in columns 73-80. For host data sets, line numbers can be in either columns 1-6 or 73-80. If the line numbers in the data set exceed the maximum allowed by $FSEDIT (32767), the system automatically renumbers the data with a smaller line number increment.

When READ completes or terminates because of an error, the number of lines transferred, or the appropriate error message, is displayed and the cursor is moved to the COMMAND INPUT field. This indicates the completion of the READ function, and you can select another option. The READ to host requires the Host Communications Facility on the System/370.

When you do a read, $FSEDIT checks for lowercase data. See "Editing Upper-Case and Lower-Case Character Data" on page UT-330 for detailed information.

Note: The system makes no attempt to verify that the data read is indeed source data. If the data is not source data, you can get unpredictable results.

Option 4 — WRITE

Option 4 (WRITE) transfers the contents of the work data set to a host/native data set. Enter a 4 in the OPTION field and the name of the target data set and volume in the DATA SET NAME and VOLUME NAME fields. $FSEDIT prompts you at the bottom of the primary option menu as follows:

```
WRITE TO dsname ON volname (Y/N)?
```

$FSEDIT - Full Screen Editor *(continued)*

If you respond Y, the contents of the work data set are saved in the data set specified in the DATA SET NAME and VOLUME NAME fields. If the data set does not exist on the volume specified, $FSEDIT creates it automatically. If you respond N, $FSEDIT resets the primary option menu and does not write the contents of the work data set to the specified data set. The WRITE to host requires the Host Communications Facility on the System/370.

Notes:

1. The WRITE option does not destroy the contents of the work data set. Therefore, you can direct WRITE to another data set to obtain a backup copy. You can also use WRITE following further editing of the work data set.

2. If you increase the contents of the work data set so that it is too big to be written back to the source data set, $FSEDIT deletes the source data set and attempts to reallocate it with enough space to save the work data set contents. If the allocation fails, the original source data set is lost. However, the work data set remains intact and can be saved in a suitable source data set.

3. If you write the work data set out to a tape, you no longer have access to the data sets previously existing on the tape.

Option 5 — SUBMIT

Option 5 (SUBMIT) injects a job (JCL and optional data) into the host job stream. The display and operation are similar to the READ and WRITE commands. The data set name entered must be the fully-qualified name of the host data set containing the JCL to be submitted. If you enter the keyword DIRECT instead of a data set name, the contents of the work data set are transferred directly into the host job stream. The SUBMIT to host requires the Host Communications Facility on the System/370.

Note: Use the DIRECT keyword only in systems with a HASP or JES2 interface.

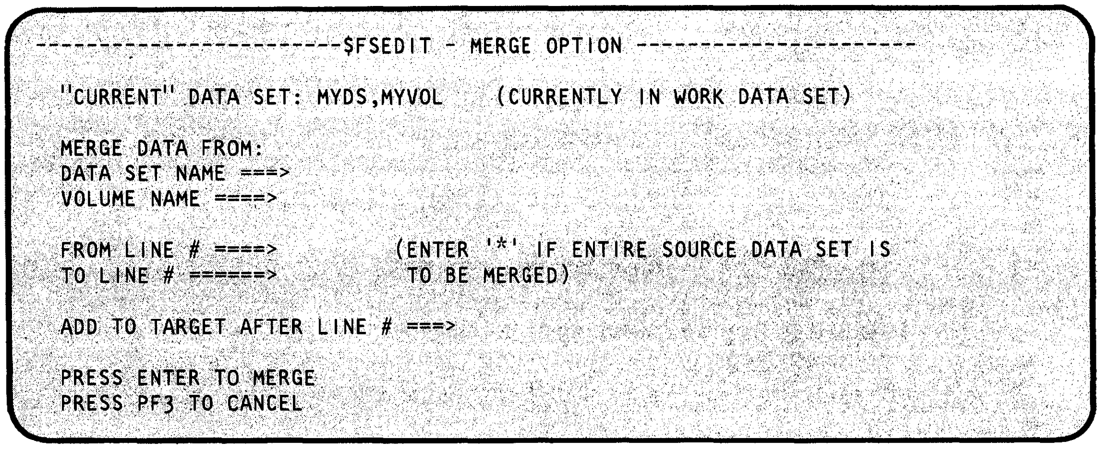
Option 6 — PRINT

Option 6 (PRINT) prints the entire contents of the work data set on $SYSPRTR. (You can end the listing at any time by pressing the attention key and typing CA.)

Option 7 — MERGE

Option 7 (MERGE) merges all, or part, of a source data set into the current edit work data set. When you select MERGE, $FSEDIT displays the following screen indicating the name of the data set that is currently in the work data set (the target).

$FSEDIT - Full Screen Editor *(continued)*



```
-----------------------$FSEDIT - MERGE OPTION ---------------------

  "CURRENT" DATA SET: MYDS,MYVOL    (CURRENTLY IN WORK DATA SET)

  MERGE DATA FROM:
  DATA SET NAME ===>
  VOLUME NAME ====>

  FROM LINE # ====>           (ENTER '*' IF ENTIRE SOURCE DATA SET IS
  TO LINE # ======>            TO BE MERGED)

  ADD TO TARGET AFTER LINE # ===>

  PRESS ENTER TO MERGE
  PRESS PF3 TO CANCEL
```

Enter the name of the data set and volume you want merged with the information in the work data set. You may type in line numbers (leading zeros are not required) or enter an asterisk if you want to merge the entire data set with the work data set. In addition, you can specify the line number in the target data set where you want the merged data to start. If you do not specify a line number, the merged data is placed at the beginning of the target data set.

The specification of an asterisk should only be used for the source data set. If all parameters are correct, the data is then read from the source data set, added to the current work data set, and the current work data set is renumbered. (If the format of the line number specification is incorrect, an error message is displayed and $FSEDIT prompts you for the data again.)

To cancel the MERGE, press the PF3 key and return to the primary menu.

Notes:

1. Once the merge has started, you must allow it to complete normally or you may get unpredictable results.

2. If you want to merge a host data set with a native data set, the line numbers in the host data set may be incompatible. To correct this incompatibility, read the host data set into the work data set and write it back out to the disk. $FSEDIT automatically resets the host data set numbers to be in columns 73-80.

Option 8 — END

Option 8 (END) ends the $FSEDIT utility.

Option 9 — HELP

Option 9 (HELP) displays tutorial text for using $FSEDIT.

$FSEDIT - Full Screen Editor *(continued)*

Primary Commands

The following primary commands are used in edit and browse mode:

CANCEL (CANC)	Cancel session and return to previous screen.
CAPS	Display data in upper or lower case.
CHANGE (C)	Change character strings.
CLEAR (CL)	Clear work data set.
COBOL	Set COBOL line numbers.
END (E)	Return to previous screen.
FIND (F)	Find a character string.
LOCATE (L)	Find specified line number.
MENU (M)	Return to primary option menu.
PS	Print current screen on $SYSPRTR.
RENUM (R)	Renumber data set.
RESET	Clear line numbers.
TABS	Set tab stops.
WRITE	Write work data set to host/native.

Primary commands are entered on line 2 of the display in the COMMAND INPUT field. You can enter all primary commands while in edit mode. In browse mode, six primary commands are recognized by $FSEDIT: LOCATE, FIND, MENU, END, CANCEL, CAPS.

You can enter most of the secondary commands in abbreviated format; for example, you can enter C instead of CHANGE. The abbreviation and full word command for each command, if applicable, are shown.

Each of the secondary commands is described on the following pages.

$FSEDIT - Full Screen Editor *(continued)*

CANC (CANCEL) — Cancel Session and Return to Previous Screen

The CANCEL command, when used in BROWSE mode, returns you to the previous screen (primary option menu or directory data set list).

The CANCEL command, when used in EDIT mode, terminates the edit without processing the screen and returns you to the previous screen (primary option menu or directory data set list).

If you create a new data set, enter data, and then enter the CANCEL command, the utility does not save any of the data you entered since you last pressed the PF key or the enter key.

If you are editing an existing data set and you add new data and then enter the cancel command, the utility saves the data as it was when you last pressed the PF or enter key. None of the new data you entered on the screen since the last pressed the PF key or enter key is saved. saved.

Syntax:

```
CANC
```

CAPS — Set Upper-Case Conversion

In BROWSE mode, the CAPS command specifies whether or not displayed data is to be converted to uppercase characters before being displayed on the screen.

In EDIT mode, the CAPS command specifies whether or not entered data is to be converted to uppercase characters when edited.

The CAPS command allows the entire work file to be converted to upper case.

Note: If you enter CAPS without specifying ON or OFF, $FSEDIT displays whether CAPS are ON or OFF. When you read in a data set and all data is upper case, the CAPS ON condition is set. If lowercase data is detected, the caps condition is set to CAPS OFF.

Syntax:

```
CAPS ON/OFF ALL

Required:  none
Defaults:  none
```

$FSEDIT - Full Screen Editor *(continued)*

Operands ***Description***

ON In BROWSE mode, all data is converted and displayed on the screen in upper case.

In EDIT mode, data is displayed on the screen exactly as it appears in the work file; for example, upper case is mixed with lower case.

If any edit data is entered and it changes the existing data, that entire line (or lines) is changed to upper case in the work file and on the screen. The following is an example of upper and lower case being read using edit mode with CAPS ON. The example has a misspelled word 'lowwercase'. If you correct the misspelling, the whole line is changed to upper case.

In this example, the first sentence shows how the line was read in.

```
This is an example of upper and lowwercase.
```

When you correct the misspelled word (lowwercase), the following occurs.

```
THIS IS AN EXAMPLE OF UPPER AND LOWER CASE.
```

OFF In BROWSE mode, all data is displayed with no conversion to upper case.

In EDIT mode, data is displayed as it is with no conversion. The data is read from the terminal with no conversion. To enter uppercase data, you must press the shift key.

Note: If $FSEDIT is being used on a terminal with only an uppercase character set, the lowercase character text data appears as blanks. If you press the enter key or any PF key, the lowercase data is lost in the work data set only.

ALL When you enter CAPS ON ALL in EDIT mode, the entire work data set is converted to upper case and CAPS is set to ON.

CAPS ON ALL has no meaning if you use it in BROWSE mode. If you use it, you will get a message.

$FSEDIT - Full Screen Editor *(continued)*

C (CHANGE) — Change Text (Edit Mode Only)

The C command changes text strings. The search for the 'text1' string proceeds until the string is found or the bottom of the data set is reached. If found, 'text1' is replaced with 'text2'. If the two text strings are not the same length, automatic shifting is performed by expanding or collapsing blank characters at the end of the line. If insufficient blanks exist for shifting right without shifting a nonblank character into column 72, the change is not made and the line is displayed with an error message in the line number field. (If you selected the ALL option, the change is ended at this point.) If the 'text1' string is not found, you are notified with an error message displayed on the top line of the screen.

Use a combination of the repeat find key (PF4) and the repeat change key (PF5) to find and change selected occurrences of a specific character string.

In CAPS OFF mode, the CHANGE command searches for the text string and when it finds it, it makes the change as entered.

In CAPS ON mode, the CHANGE command converts the text string to upper case and then searches the work file. If the text is found, the change is made and the entire line on which the text appears is converted to upper case.

Syntax:

```
CHANGE /text1/text2/option
C      /text1/text2/option

Required: /text1/text2/
Defaults: option defaults to 'NEXT'
```

Operands	***Description***
/text1/text2/	The delimiter (/) can be any alphanumeric character except blank. It is not part of, and cannot appear in, the character strings 'text1' and 'text2'. All three delimiters are required and all must be the same character. 'text1' and 'text2' can be any character string not containing the delimiter you used.

$FSEDIT - Full Screen Editor *(continued)*

option	Defines the beginning and the extent of the search. The following are valid options:

NEXT

Locate and change the next occurrence of 'text1' to 'text2'; the search starts with the first line displayed. This is the default.

FIRST

Locate and change the first occurrence of 'text1' beginning the search at the first line of data set.

ALL

Locate and change all occurrences of 'text1' beginning at the first line of the data set.

CL (CLEAR) — Clear Work Data Set (Edit Mode Only)

The CL command clears the work data set.

Syntax:

```
CLEAR
CL
```

Operands ***Description***

None

$FSEDIT - Full Screen Editor *(continued)*

COBOL — Set COBOL Line Numbers (Edit Mode Only)

The COBOL command sets line numbers in columns 1 to 6.

Syntax:

```
COBOL
CO

Required:  None
Defaults:  None
```

E (END) — Return to Previous Menu

The E command ends edit or browse mode and returns to the previous screen (primary option menu or directory data set list).

Syntax:

```
END
E
```

Operands ***Description***

None

$FSEDIT - Full Screen Editor *(continued)*

F (FIND) — Find Text String

The F command finds and displays text strings. The search proceeds until the text string is found or until the end of the data set is reached. If the string is found, automatic scrolling takes place to display the line containing the text string at the top of the data area of the display. If the string is not found, you are notified.

Use the repeat find key (PF4) to find selected occurrences of a specific character string.

In CAPS OFF mode, the FIND command searches for the text string as you entered it.

In CAPS ON mode, the FIND command converts the text string to upper case and then searches the work data set.

Syntax:

```
 FIND /text/ option
 F    /text/ option

Required: /text/
Defaults: option defaults to 'NEXT'
```

Operands	***Description***
/text/	The delimiter (/) can be any alphanumeric character except blank which does not appear within the text string. Both delimiters are required and must be the same character.
option	Defines the beginning of the search. The valid options are:

NEXT

The search starts with the first line of the current display. This is the default.

FIRST

The search starts at the first line of the data set.

$FSEDIT - Full Screen Editor *(continued)*

L (LOCATE) — Locate Line Number

The L command locates and displays the requested line number. The data set is searched for the requested line. If found, automatic scrolling takes place and the requested line is displayed at the top of the display. If the requested line number is not found, $FSEDIT issues the following message:

```
LINE NUMBER NOT FOUND
```

Syntax:

```
LOCATE line-number
L      line-number

Required: line-number
Defaults: none
```

Operands	*Description*
line-number	The number of the line to be located and displayed.

M (MENU) — Return to Primary Option Menu

The M command ends edit or browse mode and returns to the primary option menu.

Syntax:

```
MENU
M
```

Operands	*Description*
None	

$FSEDIT - Full Screen Editor *(continued)*

PS — Print the Current Screen (Edit Mode Only)

The PS command prints the currently displayed screen on the printer specified. If you do not specify a printer, the listing is directed to the $SYSPRTR.

If the screen is printed, the message SCREEN PRINTED appears in the upper right hand column of the screen. If the printer is busy, the message PRINTER BUSY appears in the upper right hand column of the screen.

Syntax:

```
PS devname

Required: none
Defaults: $SYSPRTR
```

Operands	*Description*
devname	The name of the printer where you want the display printed.

R (RENUM) — Renumber Data Set (Edit Mode Only)

The R command assigns new line numbers to each line of the data set.

Syntax:

```
RENUM first increment
R     first increment

Required: none
Defaults: first and increment default to 10 or to
        the values last used.
```

Operands	*Description*
first	The number to be assigned to the first line of the data set.
increment	The increment to be used in generating line numbers.

Note: If the number of lines in the data set is so large that the maximum line number of 32767 is exceeded, the 'first' and 'increment' values are reduced automatically until the data set can be renumbered properly.

$FSEDIT - Full Screen Editor *(continued)*

RESET — Reset Line Commands (Edit Mode Only)

The RESET command is used to reset erroneous or unwanted line commands, to reset line numbers to normal after they were replaced with ERR messages, to terminate the display of MASK and column lines, and to clear change flags. $FSEDIT issues an ERR message in the line number column when you attempt to shift left or right and there is no room on the line to do so.

Syntax:

```
RESET
```

TABS — Set Tab Stops (Edit Mode Only)

The TABS command sets a maximum of ten tab stops at the locations specified. The default setting using a newly initialized data set is 'TABS 1,10,20,30,40,50,60,70'. If you modify any default setting, the system will not support any other default settings. In such an instance, each required setting must be requested specifically. If you set tabs that are out of sequence, such as 20,30,10,40,60, $FSEDIT ignores the out of sequence tabs (in this case, 10). If you set more than 10 tabs, $FSEDIT only recognizes the first ten tabs. If you set a tab that is more than 72, the tabs will wrap around in the line number columns.

Note: The TABS command is supported on a 4978 and 4980 display terminal. See the *Operation Guide* for a description of the tab key.

Syntax:

```
TABS loc1,loc2,....,loc10

Required:  loc1
Defaults:  1,10,20,30,40,50,60,70
```

Example: Set tabs stops in columns 1, 15, and 30.

```
TABS 1,15,30
```

$FSEDIT - Full Screen Editor *(continued)*

WRITE — Save Contents of Work Data Set (Edit Mode Only)

The WRITE command saves the contents of the work data set on disk, diskette, or tape under the specified data set and volume name. If no target data set and volume is specified, the contents are saved in the data set that was read into the work data set. If the data set name is given with no specified volume name, the contents are saved in the specified data set name on the volume you are accessing currently.

Syntax:

```
WRITE data set,volume

Required:  None
Defaults:  Same data set from where the contents of the work data
        set were originally obtained.
```

Operands	*Description*
data set	The name of the data set where you want to save the contents of the work data set.
volume	The volume where the data set resides. If you do not specify a volume name, $FSEDIT assumes the data set is on the volume you are using currently.

Edit Line Commands

You can use the following edit line commands to delete, insert, duplicate, or rearrange a single line or a group of lines. They are valid only in edit mode.

)	Shift line two spaces to right.
(	Shift line two spaces to left.
>	Move a block of data two spaces to the right after the first blank.
<	Move a block of data two spaces to the left after the first blank.
A	Copy/move after.
B	Copy/move before.
C,CC	Copy line(s) of text.
COLS	Display columns.

$FSEDIT - Full Screen Editor *(continued)*

D,DD	Delete line(s) of text.
I,II	Insert line(s) of text.
MASK	Display insert mask.
M,MM	Move line(s) of text.

Single character line commands operate on a single line of data and are specified by entering the line command in the first position of the line preceding the line number.

Double character line commands operate on blocks of data and are specified by entering the block command in the first two positions on the first and last line of the block of data.

Shift Right)

Use the) symbol to shift the line two spaces to the right.

Shift Left (

Use the (symbol to shift the line two spaces to the left.

Move Right >

After the first blank, move a block of data two spaces to the right. A block of data is defined as a group of characters separated by no more than one blank.

Move Left <

After the first blank, move a block of data two spaces to the left. A block of data is defined as a group of characters separated by no more than one blank.

A (After) and B (Before)

Use the A and B commands to define the destination for a copy or move operation. 'A' defines the destination as being after the line and 'B' defines the destination as being before the line. Thus it is possible to move or copy anywhere in the data set, including before the first line or after the last line.

C (Copy Line) and CC (Copy Block)

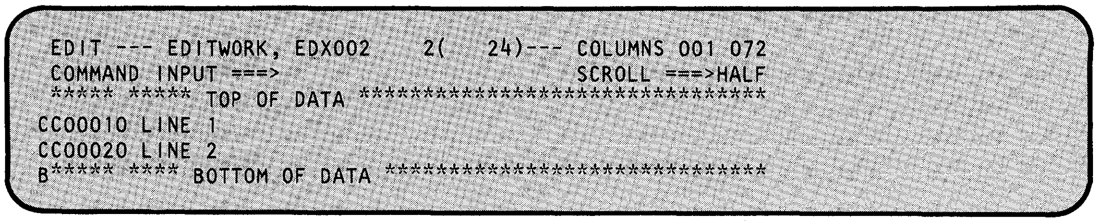
Use the C and CC command to duplicate lines of data within the data set. The block of data defined by the two CC line commands is copied to the location specified by an A or B line command. The copy operation leaves the original data intact and inserts a duplicate copy of the data at the destination specified. The copy occurs when you press the enter key after you define both the lines to be copied and the destination. The destination does not have to be on the same page of the display as the copy line command(s) and you can separate the two CC line commands also.

In CAPS OFF mode, the line(s) are copied as they are.

$FSEDIT - Full Screen Editor *(continued)*

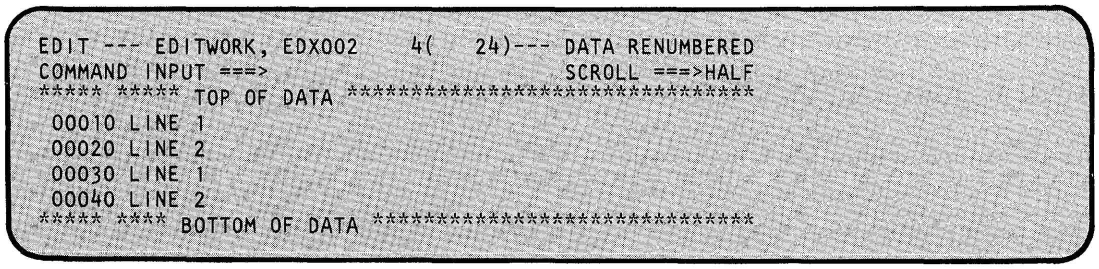
In CAPS ON mode, the copied line(s) are converted to upper case and the original data remains the same.

Example 1: Copy block.



```
 EDIT --- EDITWORK, EDX002    2(   24)--- COLUMNS 001 072
 COMMAND INPUT ===>                       SCROLL ===>HALF
 ***** ***** TOP OF DATA ********************************
CC00010 LINE 1
CC00020 LINE 2
B***** **** BOTTOM OF DATA ******************************
```

Example 2: Screen image after block copy.

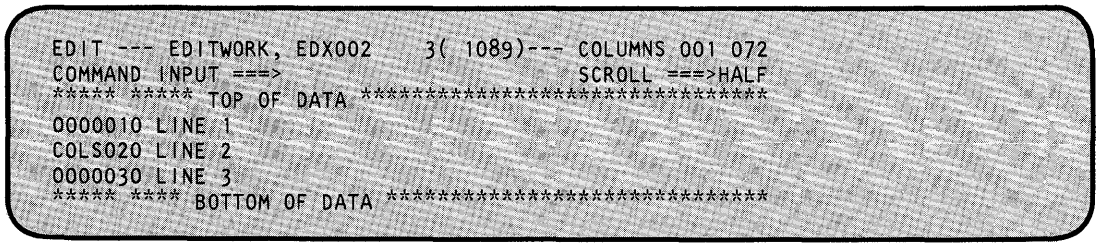


```
EDIT --- EDITWORK, EDX002    4(   24)--- DATA RENUMBERED
COMMAND INPUT ===>                       SCROLL ===>HALF
***** ***** TOP OF DATA ********************************
 00010 LINE 1
 00020 LINE 2
 00030 LINE 1
 00040 LINE 2
***** **** BOTTOM OF DATA ******************************
```

COLS — Display Columns

Use the COLS command to display a line showing column numbers. To display the column numbers, type COLS starting in the left margin of the line where the display is desired.

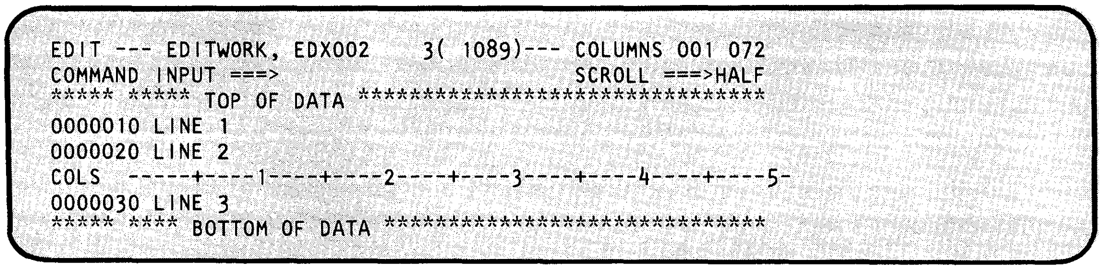
Example 1: COLS line command.



```
EDIT --- EDITWORK, EDX002    3( 1089)--- COLUMNS 001 072
COMMAND INPUT ===>                       SCROLL ===>HALF
***** ***** TOP OF DATA ********************************
0000010 LINE 1
COLS020 LINE 2
0000030 LINE 3
***** **** BOTTOM OF DATA ******************************
```

$FSEDIT - Full Screen Editor *(continued)*

Example 2: Screen image after COLS line command.

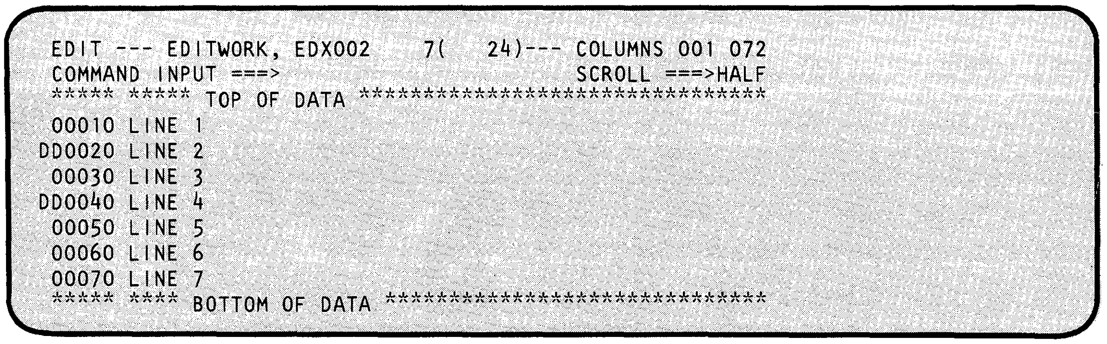


```
EDIT --- EDITWORK, EDX002    3( 1089)--- COLUMNS 001 072
COMMAND INPUT ===>                       SCROLL ===>HALF
***** ***** TOP OF DATA ********************************
0000010 LINE 1
0000020 LINE 2
COLS  -----+----1----+----2----+----3----+----4----+----5-
0000030 LINE 3
***** **** BOTTOM OF DATA ******************************
```

D (Delete Line) and DD (Delete Block)

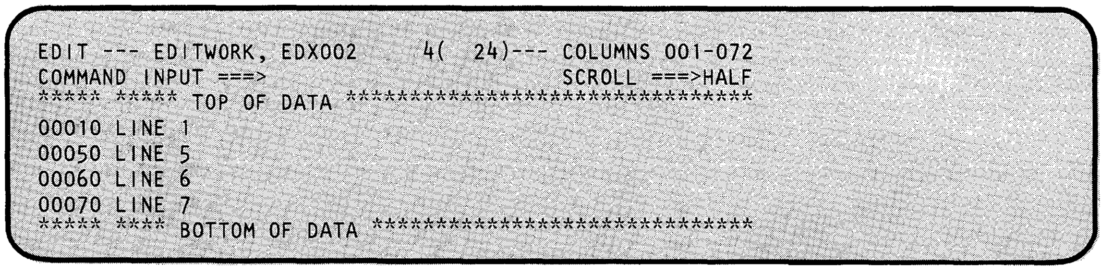
Use the D and DD commands to delete a line or a block of data. A 'D' on a line causes the line to be deleted when you press the enter key. More than one line can be deleted by entering a D on each line. The DD line commands are used to delete a block of data. The DD is entered on the first and last line of the block of data to be deleted. The first line of the block does not have to be on the same display page as the end of the block (scrolling can take place between defining the two DD lines). The block of data is deleted when you press the enter key the first time after you have specified both DD commands.

Example 1: Delete block of lines.



```
EDIT --- EDITWORK, EDX002    7(   24)--- COLUMNS 001 072
COMMAND INPUT ===>                       SCROLL ===>HALF
***** ***** TOP OF DATA ********************************
 00010 LINE 1
DD0020 LINE 2
 00030 LINE 3
DD0040 LINE 4
 00050 LINE 5
 00060 LINE 6
 00070 LINE 7
***** **** BOTTOM OF DATA ******************************
```

Example 2: Screen image after block delete.



```
EDIT --- EDITWORK, EDX002     4(  24)--- COLUMNS 001-072
COMMAND INPUT ===>                       SCROLL ===>HALF
***** ***** TOP OF DATA ********************************
00010 LINE 1
00050 LINE 5
00060 LINE 6
00070 LINE 7
***** **** BOTTOM OF DATA ******************************
```

$FSEDIT - Full Screen Editor *(continued)*

I (Insert) — Insert New Line

Use the I command to cause a new line to be inserted following the line where you enter the I command. Any information typed on the inserted line is assigned a line number and becomes part of your data when you press the enter key. If the line number assigned to the newly inserted line is equal to, or greater than, the line number of the next sequential line, all data to the end of the data set is renumbered automatically. If you don't enter any information, the inserted line is deleted automatically the next time you press the enter key.

If you enter information on the inserted line and the cursor is still on the inserted line when you press the enter key (use PF7 on the 3101), another new line is inserted automatically. This allows you to generate line after line in a continuous insert mode. The cursor is set to the first position where data appears in the following line.

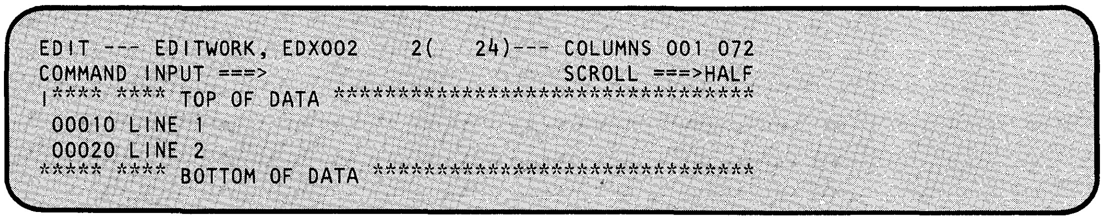
The inserted line duplicates the current value of the edit mask line. The initial value of the mask line is 72 blanks. You can change it at any time as noted in the description of the MASK command.

> **Note:** You can enter the I line command on the TOP OF DATA message line to insert a line ahead of what is currently the first line. It is typed in the first position of the TOP OF DATA line.

In CAPS OFF mode, the data is inserted in the work file as entered.

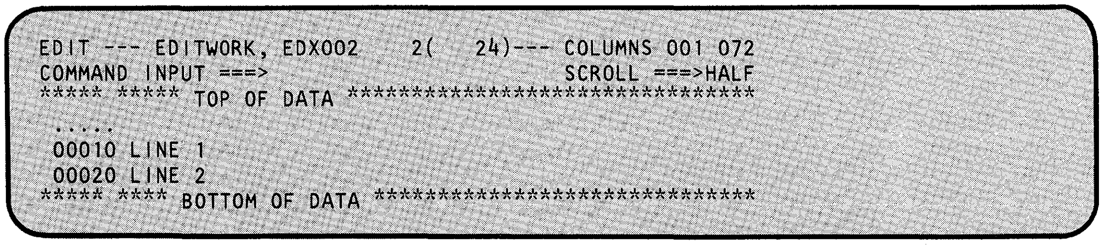
In CAPS ON mode, the data is converted to upper case before being inserted in the work file.

Example 1: Insert a line of text.



```
EDIT --- EDITWORK, EDX002    2(   24)--- COLUMNS 001 072
COMMAND INPUT ===>                        SCROLL ===>HALF
I**** **** TOP OF DATA **********************************
 00010 LINE 1
 00020 LINE 2
***** **** BOTTOM OF DATA *******************************
```

Example 2: Screen image after I line command



```
EDIT --- EDITWORK, EDX002    2(   24)--- COLUMNS 001 072
COMMAND INPUT ===>                        SCROLL ===>HALF
***** ***** TOP OF DATA *********************************
 .....
 00010 LINE 1
 00020 LINE 2
***** **** BOTTOM OF DATA *******************************
```

$FSEDIT - Full Screen Editor *(continued)*

II (Insert Block) — Insert Block of Lines

Use the II command to insert a block of new data. The line with the II command is displayed at the top of the display with twenty-one inserted lines following it. You can enter data on all twenty-one lines before you enter it with the enter key. The new data is then saved as with the I command. If all inserted lines have data entered on them and the cursor is left on the last line of the display when you press he enter key, another twenty-one lines are generated. If data is not entered on one or more of these lines, the unchanged lines are deleted and the insert mode is ended.

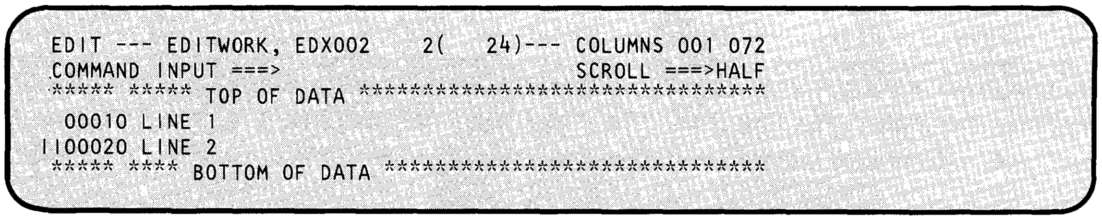
Notes:

1. The II command can be entered on the TOP OF DATA message line to insert data in front of what is now the first line. It is typed over the first two asterisks of the TOP OF DATA line.
2. The II command is different from the rest of the double character line commands. It is entered on only one line and generates a block of twenty-one blank lines.
3. The inserted lines duplicate the current value of the edit line mask.

In CAPS OFF mode, the data is inserted in the work file as you enter it.

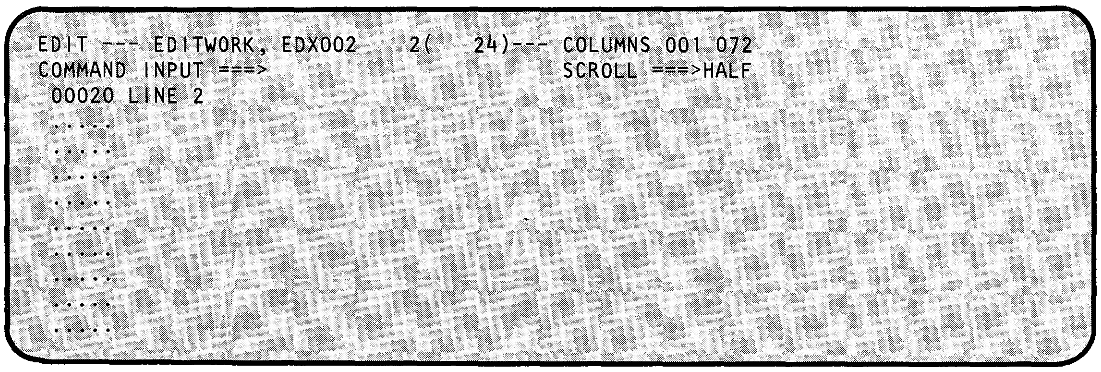
In CAPS ON mode, the data is converted to upper case before being inserted in the work file.

Example 1: Block insert line command.



```
EDIT --- EDITWORK, EDX002    2(   24)--- COLUMNS 001 072
COMMAND INPUT ===>                       SCROLL ===>HALF
***** ***** TOP OF DATA ********************************
 00010 LINE 1
II00020 LINE 2
***** **** BOTTOM OF DATA ******************************
```

Example 2: Screen image after block insert command.



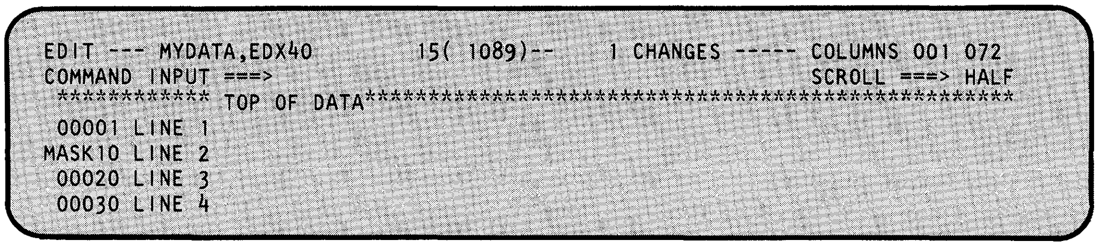
```
EDIT --- EDITWORK, EDX002    2(   24)--- COLUMNS 001 072
COMMAND INPUT ===>                       SCROLL ===>HALF
 00020 LINE 2
 .....
 .....
 .....
 .....
 .....
 .....
 .....
 .....
 .....
```

$FSEDIT - Full Screen Editor *(continued)*

MASK — Display Insert Mask

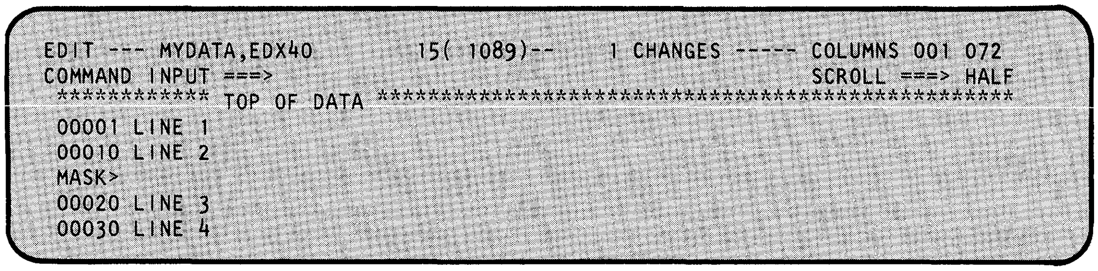
Use the MASK line command to display the insert mask which is used to prefill inserted lines. The insert mask is 72 bytes long and is initialized to all blanks the first time the work data set is used. The mask is stored in the header of the work data set. Any data filled into the mask remains in effect until you change it or you clear it by inserting blanks. The insert mask can be changed any time it is displayed by overtyping it with the desired information.

To display the insert mask, enter all four characters of the MASK line command by overtyping the first four characters of the line number. For example:



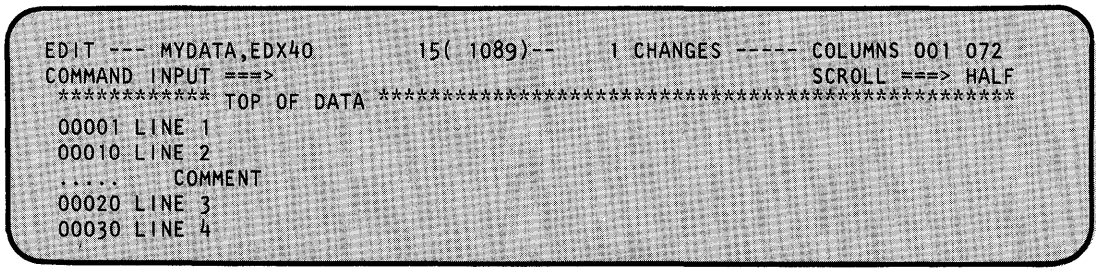
```
EDIT --- MYDATA,EDX40          15( 1089)--     1 CHANGES ----- COLUMNS 001 072
COMMAND INPUT ===>                                             SCROLL ===> HALF
 ************ TOP OF DATA*******************************************************
 00001 LINE 1
MASK10 LINE 2
 00020 LINE 3
 00030 LINE 4
```

The current mask is displayed following the line where you entered the MASK command.



```
EDIT --- MYDATA,EDX40          15( 1089)--     1 CHANGES ----- COLUMNS 001 072
COMMAND INPUT ===>                                             SCROLL ===> HALF
 ************ TOP OF DATA ******************************************************
 00001 LINE 1
 00010 LINE 2
 MASK>
 00020 LINE 3
 00030 LINE 4
```

You can replace the blank mask by overtyping it with the desired information. Once you enter the information, the mask remains in effect until you change it. Each time you enter an I to insert a line, the current mask is displayed on the insert line. In the following example, the mask has been set to the characters COMMENT.



```
EDIT --- MYDATA,EDX40          15( 1089)--     1 CHANGES ----- COLUMNS 001 072
COMMAND INPUT ===>                                             SCROLL ===> HALF
 ************ TOP OF DATA ******************************************************
 00001 LINE 1
 00010 LINE 2
 .....     COMMENT
 00020 LINE 3
 00030 LINE 4
```

To remove the mask from the screen currently being displayed, enter the RESET command on the COMMAND INPUT line. You cannot remove the mask using the CLEAR command.

$FSEDIT - Full Screen Editor *(continued)*

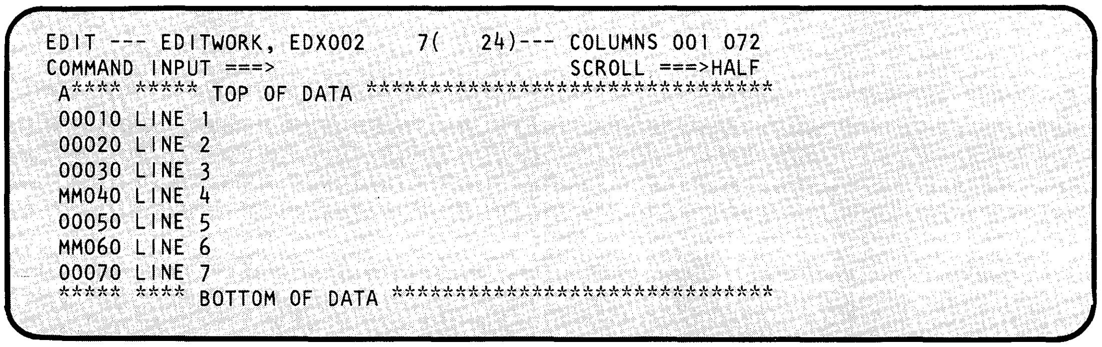
M (Move Line) and MM (Move Block)

Use the M and MM commands to move a line or block of lines from one location to another. When you enter an M line command, a single line is moved to the location specified by an A or B line command. The MM line command causes the block of data defined by the two MM line commands to be moved to the location specified by an A or B line command. The moved lines are removed from their original location and the entire data set may be renumbered after the move. The move occurs when you press the enter key the first time after you define both the lines to be moved and the destination. The destination does not have to be on the same page of the display as the move line command(s) and you can separate the two MM line commands also.

In CAPS OFF mode, the data is moved and remains the same.

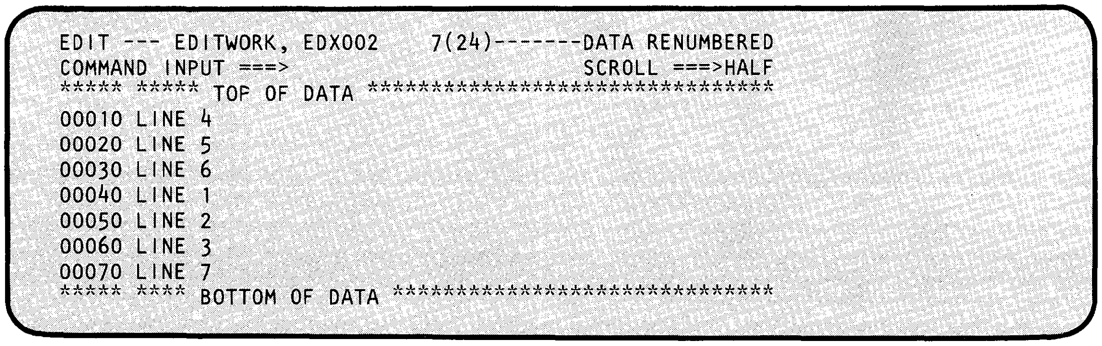
In CAPS ON mode, the data is converted to upper case before being moved.

Example 1: Move block of lines.



```
EDIT --- EDITWORK, EDX002    7(   24)--- COLUMNS 001 072
COMMAND INPUT ===>                       SCROLL ===>HALF
 A**** ***** TOP OF DATA ********************************
 00010 LINE 1
 00020 LINE 2
 00030 LINE 3
 MM040 LINE 4
 00050 LINE 5
 MM060 LINE 6
 00070 LINE 7
 ***** **** BOTTOM OF DATA ******************************
```

Example 2: Screen image after block move.



```
EDIT --- EDITWORK, EDX002    7(24)-------DATA RENUMBERED
COMMAND INPUT ===>                      SCROLL ===>HALF
***** ***** TOP OF DATA ********************************
00010 LINE 4
00020 LINE 5
00030 LINE 6
00040 LINE 1
00050 LINE 2
00060 LINE 3
00070 LINE 7
***** **** BOTTOM OF DATA ******************************
```

$GPIBUT1

$GPIBUT1 Utility

The $GPIBUT1 utility enables you to control and transfer data to and from GPIB devices interactively. You can use this utility as a diagnostic tool, also, to check out the application program interface and the attached devices.

Invoking $GPIBUT1

You invoke $GPIBUT1 with the $L operator command or option 4.9 of the session manager.

```
> $L $GPIBUT1
 LOADING GPIBUT1    45P,00:40:27:, LP= 9200, PART=1

 $GPIBUT1 - GPIB UTILITY

 COMMAND (?):
```

$GPIBUT1 Commands

To display the $GPIBUT1 commands at your terminal, enter a question mark in response to the prompting message, COMMAND (?):.

```
COMMAND(?): ?

CH   - DEFINE END CHARACTER
CP   - CHANGE GPIB PARTITION
DD   - DEFINE DEVICE
EN   - END THE PROGRAM
GP   - GPIB CONTROL
LDCB - LIST DEVICE CONTROL BLOCK
RE   - READ DATA
RS   - RESET I/O ADAPTER
ST   - READ ERROR STATUS
SU   - SUSPEND PROGRAM
WR   - WRITE DATA
'ATTN - PGPIB' TO POST
'ATTN - GPRESUME' TO RESUME PROGRAM

COMMAND(?):
```

If a $GPIBUT1 command fails, use the attention list command PGPIB to terminate the failing operation. If you use PGPIB, you must issue an RS command (or RSET if GP subcommands are used) to reset the adapter.

$GPIBUT1 Utility *(continued)*

CH — Define End Character

Use the CH command to define or change the ending character that is added to output data.

```
COMMAND(?):  CH

CHARACTER TO BE APPENDED TO OUTPUT DATA -- NOW IS NONE

1 = CARRIAGE RETURN
2 = LINE FEED
3 = END OF TEXT
4 = USER SPECIFIED HEX BYTE
5 = NONE

SELECT CODE:  3

  END CHARACTER IS NOW ETX

COMMAND(?):
```

CP — Change GPIB Partition

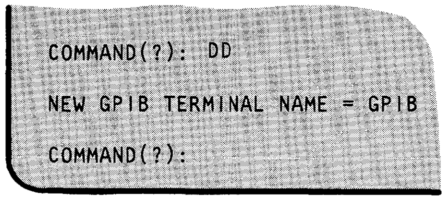
Use the CP command to change the partition to which the GPIB adapter is connected. The partition is initially defined at system generation.

```
COMMAND(?):  CP 2

  PARTITION CHANGED TO 2

COMMAND(?):  CP

  PARTITION NUMBER (NOW IS 2):

  PARTITION NUMBER NOT CHANGED

COMMAND(?):
```

$GPIBUT1 Utility *(continued)*

DD — Define Device

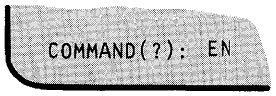
Use the DD command to prompt for the name of the GPIB adapter. This name is specified in the TERMINAL configuration statement. The name specified is used for all enqueues of the adapter until you issue another DD command.



```
COMMAND(?): DD

NEW GPIB TERMINAL NAME = GPIB

COMMAND(?):
```

EN — End The Program

Use the EN command to end the $GPIBUT1 utility.



```
COMMAND(?): EN
```

GP — GPIB Control

Use the GP command to enter the GPIB bus command options that can be specified using the TERMCTRL instruction. These are described in the *Language Reference*.

When you enter GP and follow it by a bus command, $GPIBUT1 prompts for additional data, depending upon the specific command. For example, the CON (configure) command requires both configuration and programming data. For the REN (remote enable) command, you must include a list of GPIB device addresses.

Where appropriate, $GPIBUT1 performs PRINTEXT/READTEXT operations as part of the execution of a GP command, inserting delimiters as needed. In some cases, one delimiter is a user-defined end character. The end character can be defined by the CH command.

$GPIBUT1 Utility *(continued)*

```
COMMAND(?):  GP

GPIB COMMAND(?):  CON

OPTION(SE,EOS,TO,EOI):  TO              (timer override)
OPTION(SE,EOS,TO,EOI):

CONFIGURATION DATA:  ?U%                (S/1 talks, plotter listens)
PROGRAMMING DATA (OR NONE):  IN         (initialize plotter)
PROGRAMMING DATA (OR NONE):  OE;        (output plotter error)
PROGRAMMING DATA (OR NONE):

CONFIGURATION DATA:  ?E5                (plotter talks, S/1 listens)
PROGRAMMING DATA (OR NONE):  NONE       (no data)
PROGRAMMING DATA (OR NONE):

CONFIGURATION.DATA:

GPIB COMMAND(?): READ
OPTION(SE,EOS,TO,EOI):  SE              (suppress exceptions)
                          WARNING - EOS OR EOI REQUIRED ...
OPTION(SE,EOS,TO,EOI):  EOS             (end of string character)
                          WARNING - SE MAY BE NEEDED ...
  EOS BYTE (HEX):  OD                          (X'OD')
OPTION(SE,EOS,TO,EOI):
TRANSLATE INPUT?  Y

HOW MANY CHARACTERS (MAX=DEFAULT=80):  (let default)
VALUE DEFAULTED TO 80

0                                       (error code=0)

COMMAND(?):
```

LDCB — List Device Control Block

Use the LDCB command to list the contents of the current GPIB device control block (DCB). The DCB describes the last GPIB operation performed. However, the information provided may require that you use the GPIB Adapter manual. The items listed include:

- Address of the GPIB terminal control block (CCB)
- Address of the GPIB device control block (DCB)
- Status of the DCB control word, specifically:
  - Cycle-steal status key (that is, the address space of the data buffer)
  - GPIB operation mnemonic (for an undefined operation, '****')
  - Status of the chaining, input, suppress exception (SE), end of string (EOS), timer override (TO), and end of identify (EOI) bits, if they are set

$GPIBUT1 Utility *(continued)*

- End of string character
- Address of the residual status block (RSB)
- Chain address
- Byte count for the data transfer
- Address of the data buffer
- Contents of the data buffer, expressed as:
  - A string of hexadecimal words
  - EBCDIC characters
  - ASCII characters.

The DCB is checked for certain error conditions, including:

- DCB words two or three not equal to zero
- RSB address not equal to zero, and suppress exception set
- Chain address nonzero and chaining bit set.

If the byte count is odd, the last byte in the string of hex words is not part of the buffer and should be disregarded. Because the buffer data can be either EBCDIC or ASCII, depending on the application, it is displayed in both character codes. In most cases, the ASCII data that is displayed will be accurate. An inappropriate translation is displayed as a blank line.

$GPIBUT1 Utility *(continued)*

The following example illustrates a DCB used in the execution of:

```
TERMCTRL GPIB,CON,TO
PRINTEXT '?U%":@'
```

```
COMMAND (?):  LDCB

DISPLAY OF DCB FOR GPIB TERMINAL GPIB

GPIB CCB AT ADDRESS 1058
GPIB DCB AT ADDRESS OFFE

CONTROL WORD
  CYCLE STEAL KEY IS 0
  TIMER OVERRIDE SET
  DEVICE OPERATION IS CON

BYTE COUNT IS   5
DATA ADDRESS IS 1102

DATA IN HEX FORMAT IS:

3F55 2522 2C00

DATA INTERPRETED AS EBCDIC IS:

?U%":

COMMAND (?):
```

RE — Read Data

Use the RE command to read data from the GPIB adapter. You can also specify GPIB options (TERMCTRL functions), translation, and the number of characters to be read.

```
COMMAND(?):  READ
OPTION(SE,EOS,TO,EOI):  SE            (suppress exceptions)
                         WARNING - EOS OR EOI REQUIRED ...
OPTION(SE,EOS,TO,EOI):  EOS           (end of string character)
                         WARNING - SE MAY BE NEEDED ...
  EOS BYTE (HEX):  0D                 (X'0D')
OPTION(SE,EOS,TO,EOI):
TRANSLATE INPUT?  Y

HOW MANY CHARACTERS (MAX=DEFAULT=80):    (let default)
VALUE DEFAULTED TO 80

0                                        (error code=0)

COMMAND(?):
```

$GPIBUT1 Utility *(continued)*

RS — Reset I/O Adapter

Use the RS command to issue a device reset to the adapter. Any pending interrupt or busy condition is cleared when this command is executed.

```
COMMAND(?):  RS

  RESET ADAPTER?  Y

COMMAND(?):
```

ST — Read Error Status

Use the ST command to display the status information contained in the adapter cycle steal status words and the residual status block (RSB).

```
COMMAND(?):  ST

READ STATUS?  Y

CYCLE STEAL STATUS BLOCK (HEX)

  RESIDUAL ADDRESS =             B45A
  RESIDUAL BYTE COUNT =          0050
  (RESERVED) =                   0000
  (RESERVED) =                   0000
  ERROR STATUS =                 8000
  BUS STATUS (AFTER POWER ON) =  0008
  BUS STATUS (CURRENT) =         000A
  SPE DEVICE ADDRESS =           0000
  DCB SPECIFICATION CHECK =      0000
  (RESERVED) =                   0000
  DCB ADDRESS =                  1238

RESIDUAL STATUS BLOCK (HEX)

  RESIDUAL BYTE COUNT = 0000
  RSB FLAGS =           0000
  (RESERVED) =          0000
  (RESERVED) =          0000
  (RESERVED) =          0000  (error status indicates time-out
                               waiting to receive data)
RESET ADAPTER?  Y

NO DATA RECEIVED

COMMAND(?):
```

$GPIBUT1 Utility *(continued)*

SU — Suspend $GPIBUT1

Use the SU command to suspend the operation of $GPIBUT1 until you tell it to resume using GPRESUME. This enables you to run $GPIBUT1 concurrently with a GPIB application from the same terminal.

```
COMMAND(?):  SU

$GPIBUT1 SUSPENDED
```

WR — Write Data

Use the WR command to write data to the GPIB adapter. You can specify GPIB options (TERMCTRL functions) and translation.

```
COMMAND(?):  WR

OPTION(SE,EOS,TO,EOI):

  WRITE HEX DATA?  N

  ENTER TEXT:  IN;                    (character data)

  IS THE DATA OK?  Y

COMMAND(?):  WR

OPTION(SE,EOS,TO,EOI):

  WRITE HEX DATA?  Y

  ENTER HEX WORDS:  0A0D 0102         (hex data)

  IS THE DATA OK?  Y

COMMAND(?):
```

$GPIBUT1 Utility *(continued)*

PGPIB — Post GPIB Operation Completion

Sometimes a GPIB operation waits indefinitely, such as when a device fails to respond to an operation in which timer override (TO) is specified. Use the PGPIB attention command to complete the operation; it cancels the operation by simulating its completion. However, after a PGPIB, the GPIB adapter is still in a busy state, so you must reset it.

```
GPIB COMMAND (?):  READ

OPTION (SE,EOS,TO,EOI):  TO

TRANSLATE INPUT?  Y

HOW MANY CHARACTERS (MAX=DEFAULT=80):

VALUE DEFAULTED TO 80              (user presses attention key)

> PGPIB

DATA RECEIVED:

GPIB COMMAND (?):  RSET

  RESET ADAPTER?  Y
```

GPRESUME — Resume $GPIBUT1 Operation

If you suspended $GPIBUT1 using the SU command, use the GPRESUME attention command to resume it.

```
(press attention key)

> GPRESUME

$GPIBUT1 RESUMED

COMMAND (?):
```

$GPIBUT1 Example

The following example shows many of the $GPIBUT1 utility operations. The attached device is a plotter.

$GPIBUT1 Utility *(continued)*

```
(Press attention key)

> $CP 2                      (run in partition 2)
> $L $GPIBUT1
LOADING $GPIBUT1    45P,00:40:27, LP= 0000, PART=2

$GPIBUT1 - GPIB UTILITY

USING GPIB TERMINAL GPIB1

COMMAND(?):  DD

NEW GPIB TERMINAL NAME = GPIB3  (invalid GPIB name)

 GPIB3 IS NOT A GPIB TERMINAL

RETRY?  Y

NEW GPIB TERMINAL NAME = GPIB

COMMAND (?):  CP 2            (so that SRQs will be received)
 PARTITION CHANGED TO 2

COMMAND (?):  CH

CHARACTER TO BE APPENDED TO OUTPUT DATA -- NOW IS NONE

 1 = CARRIAGE RETURN
 2 = LINE FEED
 3 = END OF TEXT
 4 = USER SPECIFIED HEX BYTE
 5 = NONE

SELECT CODE:  3

 END CHARACTER NOW IS ETX
```

Figure 20 (Part 1 of 5). $GPIBUT1 example

$GPIBUT1 Utility *(continued)*

```
COMMAND (?):  CH
CHARACTER TO BE APPENDED TO OUTPUT DATA -- NOW IS ETX

 1 = CARRIAGE RETURN
 2 = LINE FEED
 3 = END OF TEXT
 4 = USER SPECIFIED HEX BYTE
 5 = NONE

SELECT CODE:  5

  END CHARACTER NOW IS NONE
COMMAND (?):  GP

GPIB COMMAND (?):  CON
OPTION(SE,EOS,TO,EOI):  TO          (timer override)
OPTION(SE,EOS,TO,EOI):

CONFIGURATION DATA:  ?U%            (S/1 talks, plotter listens)
PROGRAMMING DATA (OR NONE):  IN;    (initialize plotter)
PROGRAMMING DATA (OR NONE):  OE;    (output plotter error)
PROGRAMMING DATA (OR NONE):

CONFIGURATION DATA:  ?E5            (plotter talks, S/1 listens)
PROGRAMMING DATA (OR NONE):  NONE   (no data)
PROGRAMMING DATA (OR NONE):

CONFIGURATION DATA:

GPIB COMMAND (?):  READ
OPTION(SE,EOS,TO,EOI):  SE          (suppress exceptions)
                         WARNING - EOS OR EOI REQUIRED ...
OPTION(SE,EOS,TO,EOI):  EOS         (end of string character)
                         WARNING - SE MAY BE NEEDED ...
  EOS BYTE (HEX):  0D               (X'0D')
OPTION(SE,EOS,TO,EOI):
TRANSLATE INPUT?  Y

HOW MANY CHARACTERS (MAX=DEFAULT=80):       (let default)
VALUE DEFAULTED TO 80

0                                           (error code=0)
```

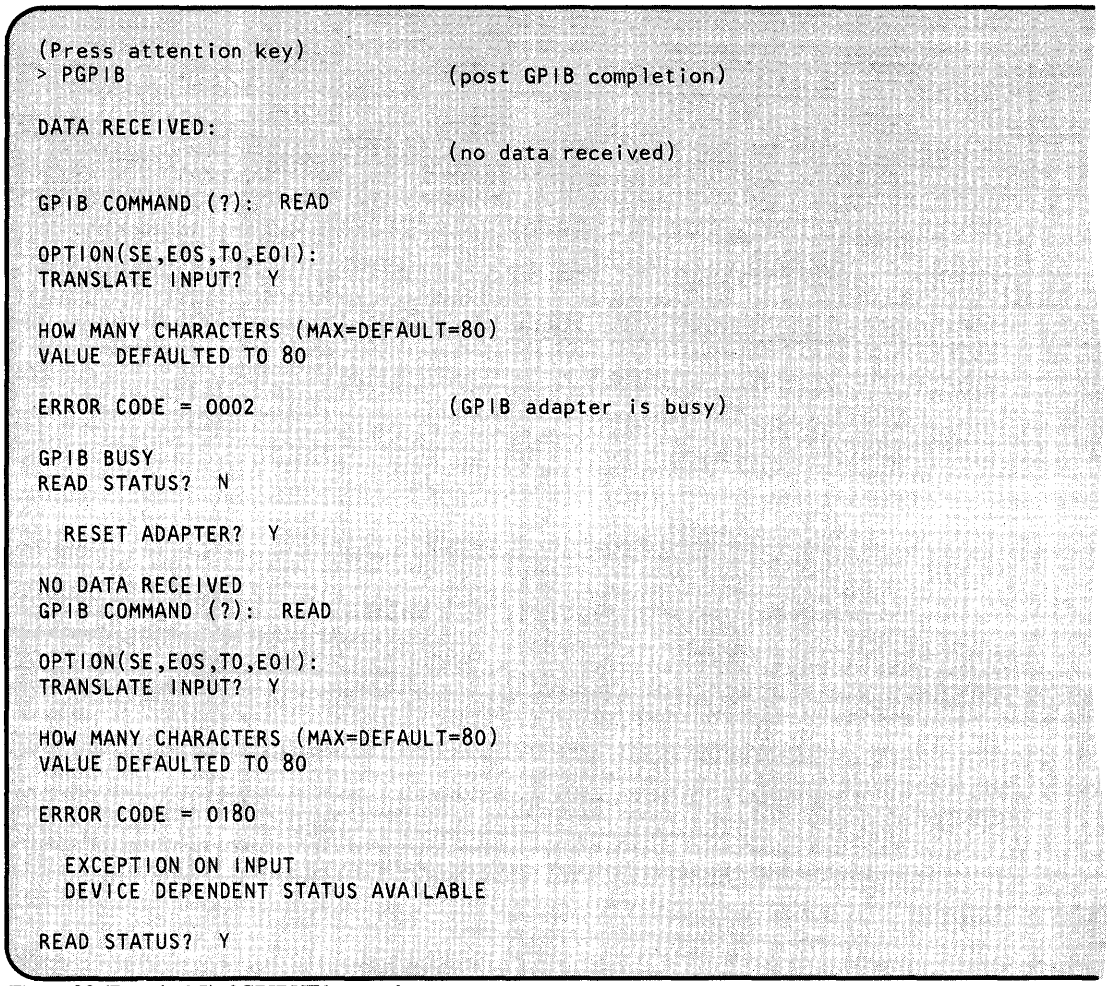
Figure 20 (Part 2 of 5). $GPIBUT1 example

$GPIBUT1 Utility *(continued)*

```
GPIB COMMAND(?):  CON

OPTION(SE,EOS,TO,EOI):

CONFIGURATION DATA:  ?U%
PROGRAMMING DATA (OR NONE):  IN;              (initialize plotter)
PROGRAMMING DATA (OR NONE):  IM223,32,0;      (interrupt on error)
PROGRAMMING DATA (OR NONE):  XX;              (invalid command)

******  SRQ RECEIVED  ******                  (SRQ interrupt)

PROGRAMMING DATA (OR NONE):

CONFIGURATION DATA:
GPIB COMMAND (?):  SPE                        (serial poll enable)

OPTION(SE,EOS,TO,EOI):

  TALKER ADDRESS LIST:  E                     (plotter talk address)
  IS THE DATA OK?  Y

GPIB COMMAND (?):  SPL                        (read serial poll)
WARNING - SPE MUST HAVE JUST BEEN EXECUTED

TRANSLATE INPUT?  N                           (no translation)

DATA RECEIVED:
 4578                                         (plotter address, status)

WARNING - AN SPD MAY NOW BE REQUIRED

GPIB COMMAND (?):  SPD                        (serial poll disable)
OPTION(SE,EOS,TO,EOI):

GPIB COMMAND (?):  READ
OPTION(SE,EOS,TO,EOI):  TO                    (timer override)

TRANSLATE INPUT?  Y                           (translate)
HOW MANY CHARACTERS (MAX=DEFAULT=80):

VALUE DEFAULTED TO 80
                              (read is in indefinite wait)
```

Figure 20 (Part 3 of 5). $GPIBUT1 example

$GPIBUT1 Utility *(continued)*



```
(Press attention key)
> PGPIB                          (post GPIB completion)

DATA RECEIVED:
                                 (no data received)

GPIB COMMAND (?):  READ

OPTION(SE,EOS,TO,EOI):
TRANSLATE INPUT?  Y

HOW MANY CHARACTERS (MAX=DEFAULT=80)
VALUE DEFAULTED TO 80

ERROR CODE = 0002                (GPIB adapter is busy)

GPIB BUSY
READ STATUS?  N

  RESET ADAPTER?  Y

NO DATA RECEIVED
GPIB COMMAND (?):  READ

OPTION(SE,EOS,TO,EOI):
TRANSLATE INPUT?  Y

HOW MANY CHARACTERS (MAX=DEFAULT=80)
VALUE DEFAULTED TO 80

ERROR CODE = 0180

  EXCEPTION ON INPUT
  DEVICE DEPENDENT STATUS AVAILABLE

READ STATUS?  Y
```

Figure 20 (Part 4 of 5). $GPIBUT1 example

$GPIBUT1 Utility *(continued)*

```
CYCLE STEAL STATUS BLOCK (HEX)

  RESIDUAL ADDRESS =              B45A
  RESIDUAL BYTE COUNT =           0050
  (RESERVED) =                    0000
  (RESERVED) =                    0000
  ERROR STATUS =                  8000
  BUS STATUS (AFTER POWER ON) =   0008
  BUS STATUS (CURRENT) =          000A
  SPE DEVICE ADDRESS =            0000
  DCB SPECIFICATION CHECK =       0000
  (RESERVED) =                    0000
  DCB ADDRESS =                   1238
RESIDUAL STATUS BLOCK (HEX)

  RESIDUAL BYTE COUNT = 0000
  RSB FLAGS =           0000
  (RESERVED) =          0000
  (RESERVED) =          0000
  (RESERVED) =          0000  (error status indicates time-out
                               waiting to receive data)
  RESET ADAPTER?   Y

NO DATA RECEIVED

GPIB COMMAND (?):   END
```

Figure 20 (Part 5 of 5). $GPIBUT1 example

$HCFUT1 - Interact with Host Communications Facility

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

- Read a data set from the host (READDATA, READ80, READOBJ).
- Write a data set to the host (WRITE).
- Submit a job to the host (SUBMIT).
- Status - Set, fetch, and release records in the system status data set.

Invoking $HCFUT1

You invoke $HCFUT1 with the $L operator command or option 8.8 of the session manager.

$HCFUT1 Commands

To display the $HCFUT1 commands at your terminal, enter a question mark in response to the prompting message COMMAND (?):.

```
COMMAND (?): ?

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

COMMAND (?):
```

After $HCFUT1 displays the commands, it prompts you with COMMAND (?): again. Then you can respond with the command of your choice (for example, SU).

Notes:

1. See 'Host Data Set Naming Conventions' and 'Host Data Set Characteristics' in *Communications Guide.*
2. See 'System Status Data Set' in *Communications Guide.* Appendix B of the *IBM Series/1 Host Communications Facility Program Description and Operation Manual*, SH20-1819, contains more details on its use.

$HCFUT1 - Interact with Host Communications Facility *(continued)*

3. The Host Communications Facility IUP, program number 5796-PGH, is required on the host System/370.

4. Host Communications Facility must be installed and configured on the Series/1.

READDATA — Transfer data set from host to Series/1

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

Three items of control information you must specify at execution time 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 $HCFUT1
LOADING $HCFUT1     8P,08.15.30, LP=4B00
WORKFILE(NAME,VOLUME):  DATAFIL2,EDX001

COMMAND (?):   READDATA
NO. OF RECORDS TO READ(0=ALL):   0
DSNAME:  S1.EDX.TESTIN(DATA)
END AFTER          40 RECORDS

COMMAND (?):
```

READ80 and READOBJ — Transfer Records from Host to Series/1

READ80 and READOBJ 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.

$HCFUT1 - Interact with Host Communications Facility *(continued)*

READOBJ stores three 80-byte records in the first 240 bytes of each disk record. This format is compatible with object modules produced by any of the assembler programs. It is also the format required for input to $EDXLINK and is one of the formats accepted by $UPDATE. READOBJ is normally used to transfer the output object module of a host assembly to the Series/1 for processing by $EDXLINK or $UPDATE.

You invoke both READ80 and READOBJ in a manner similar to READDATA.

Status Commands (SE, FE and REL)

The status commands are used to perform, from a terminal, the SET, FETCH, and RELEASE functions on the system status data set. See *Messages and Codes* 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 records have been added.

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

The following is an example 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
KEY = MISSING1
STATUS =        900
COMMAND (?):
```

SUBMIT (SU) — Submit Job to Host Job Stream

SUBMIT causes a job to be submitted to the host job stream. See *Communications Guide* for information about the requirements for host data sets.

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 SU to submit the data set 'S1.EDX.TESTSUB.CNTL'.

$HCFUT1 - Interact with Host Communications Facility *(continued)*

```
COMMAND (?):  SU
DSNAME:   S1.EDX.TESTSUB.CNTL
JOB SUBMITTED
ANOTHER JOB?  N

COMMAND (?):
```

WRITE (WR) — Transfer Data Set from Series/1 to Host

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

Three items of control information you must specify at execution time 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 consists of up to 44 characters, or 54 characters for 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
LOADING $HCFUT1     8P,08.15.20, LP=4B00, PART=2

COMMAND (?):  WR
NO. OF RECORDS TO WRITE(0=ALL):   28
DSNAME:  S1.EDX.TESTOUT.DATA
END AFTER 28

COMMAND (?):
```

$HCFUT1 - Interact with Host Communications Facility *(continued)*

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 *Messages and Codes*.

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

$HXUT1 - H-Exchange Utility

$HXUT1 allows you to perform different functions for H-exchange data sets and volumes. You can transfer data contained on $HXUT1 diskettes from one system to another providing the target system can read diskettes formatted for Standards for Information Interchange.

Volume-oriented functions consist of the following:

- CV - Change to another H-exchange volume.
- IV - Initialize volume.
- LA - List contents of H-exchange volume.
- LI - Redirect listings to another terminal.
- LS - List space in H-exchange volume.
- RE - Rename an H-exchange volume.
- UV - Update H-exchange volume label.

Data-set-oriented functions consist of the following commands:

- AL - Allocate data set.
- DE - Delete data set.
- UD - Update H-exchange data set label by name.
- UH - Update H-exchange data set label by number.

You can copy data between H-exchange and EDX data sets with the following commands:

- RX - Read H-exchange data set into EDX data set.
- WX - Write H-exchange data set from EDX data set.

Note: For information on copying basic exchange diskettes see "$COPY - Copy Data Set" on page UT-72.

$HXUT1 - H-Exchange Utility *(continued)*

Invoking $HXUT1

You invoke the H-exchange utility with the $L operator command or option 3.11 of the session manager. Once invoked, $HXUT1 prompts you for the name of a diskette 2D volume in order to start the program.

```
> $L $HXUT1
LOADING $HXUT1      72P,09:18:62, LP=2300, PART=1

$HXUT1 - H EXCHANGE UTILITY

VOLUME NAME =
```

If the volume name you enter is not an H-exchange diskette, $HXUT1 asks if you wish to retry.

```
 TRY AGAIN (Y/N):
```

Respond Y and $HXUT1 again prompts you for the name of the volume.

```
 VOLUME NAME =
```

If you want to exit the program, press the enter key in response to VOLUME NAME=, and enter N in response to the TRY AGAIN (Y/N)? prompt.

```
 VOLUME NAME =

 TRY AGAIN (Y/N)?  N

 $HXUT1 ENDED AT  09:18:23
```

$HXUT1 - H-Exchange Utility *(continued)*

$HXUT1 Commands

To display the $HXUT1 commands at your terminal, enter a question mark in response to the prompting message COMMAND (?):.

```
COMMAND (?):   ?

AL - ALLOCATE DATA SET
CV - CHANGE VOLUME
DE - DELETE DATA SET
IV - INITIALIZE VOLUME
LA - LIST ALL DATA SETS
LI - CHANGE LISTING DEVICE
LS - LIST AVAILABLE SPACE
RE - RENAME VOLUME
RX - READ H-EXCHANGE DATA SET INTO EDX DATA SET
UD - UPDATE DATA SET LABEL BY NAME
UH - UPDATE DATA SET LABEL BY NUMBER
UV - UPDATE VOLUME LABEL
WX - WRITE H-EXCHANGE DATA SET FROM EDX DATA SET
EN - END PROGRAM

COMMAND (?):
```

After $HXUT1 displays the commands, you are prompted again with COMMAND (?):. Then you can respond with the command of your choice (for example, AL).

Using the H-exchange utility

Commands that require parameter information prompt you for the parameters. Advance input is acceptable, except where data is being destroyed. In those commands which alter a value, the current value is displayed prior to the entry of the new value. Do not enter a new value if the current value is to be saved; just press the enter key.

You can direct certain listings produced by this utility to another terminal or printer. You can do this with the LI command. The LI command enables you to list the volume contents, volume, or data set labels.

Data Set Copying Considerations

The copy commands support partial copies. You can select a portion of the data set to be copied by specifying the starting and ending record numbers. These record numbers are relative to the overall data set. In the case of a multivolume copy, the numbers are relative to all preceding segments of the data set, not the current segment. Segments are the portions of multivolume data sets that span diskettes. If one data set is copied completely to another, the end of data set pointer is set to reflect the last record copied. Once a data set on a multivolume data set is allocated, the continued/lst indicators are not modified on subsequent use of the data set.

$HXUT1 - H-Exchange Utility *(continued)*

Naming Conventions

$HXUT1 supports a naming convention for volumes containing segments of a data set. If the volume name for the current segment of a data set is XXXXnn (where nn is "01" through "98"), then the copy command searches for a volume name of the form XXXXmm, where mm=nn+1. For example, if the volume name for the current segment of a data set is FILE01, the copy command searches for a volume name of FILE02.

AL — Allocate H-Exchange Data Set

Use the AL command to allocate a single volume H-exchange data set. You must enter the name and size of the data set.

```
COMMAND (?): AL

DATA SET NAME: CUSTFILE

NUMBER OF RECORDS: 100

CUSTFILE ALLOCATED, SIZE IS 100 RECORDS
```

$HXUT1 - H-Exchange Utility *(continued)*

When there is insufficient space in the volume to allocate the data set, $HXUT1 informs you. You then can allocate the data set at the size of the largest remaining free space in the volume.

```
COMMAND (?):  AL

DATA SET NAME:  CUSTFILE

NUMBER OF RECORDS:  2000

INSUFFICIENT SPACE TO ALLOCATE DATA SET
THE LARGEST EXTENT AVAILABLE CONTAINS 1500 RECORDS
ALLOCATE DATA SET FOR THIS SIZE (Y/N)?  Y

CUSTFILE ALLOCATED, SIZE IS 1500 RECORDS
```

To allocate a data set which already exists, the following rules apply to the existing data set. If the existing data set is:

- Write protected - the command is terminated.
- Larger than or equal to your request - it is used to satisfy your request.
- Too small to satisfy your request - you are asked if the data set should be saved.

```
CUSTFILE ALREADY EXISTS, SIZE IS 100 RECORDS
SAVE IT? (Y/N)
```

Note: If the data set is saved, the command is terminated. If the data set should not be saved, it is deleted, and the new data set is allocated elsewhere in the volume.

$HXUT1 - H-Exchange Utility *(continued)*

CV — Change to Another H-Exchange Volume

Use the CV command to change the H-exchange volume. (This affects all commands except for the copy commands.)

```
COMMAND (?): CV

VOLUME NAME (NOW IS 'ENTIEV') =  VOLCAN
```

DE — Delete Data Set

Use the DE command to delete a H-exchange data set, if it is not write protected. You must enter the name of the data set.

```
COMMAND (?): DE

DATASET NAME: ACCTREC

ARE YOU SURE (Y/N)? Y

ACCTREC DELETED

COMMAND (?):
```

IV — Initialize Volume

Use the IV command to initialize a diskette 2D for processing as a H-exchange volume. The diskette must have been formatted by $DASDI to be compatible with interchange standards and at 256 bytes per sector.

```
COMMAND (?): IV

               *** WARNING ***

INITIALIZATION WILL DELETE ALL EXISTING DATA SETS --
CONTINUE (Y/N)? Y

INITIALIZATION COMPLETE

COMMAND (?):
```

$HXUT1 - H-Exchange Utility *(continued)*

LA — List Contents of H-Exchange Volume

Use the LA command to list the contents of an H-exchange volume. The list operation lists the names of the data sets contained on an H exchange diskette. The location and size are also listed, both in EDX and CCHSS format. The listing also indicates if the data set is continued on another volume or is the last in a sequence. In both cases, the sequence number is displayed.

If listings are being redirected to another terminal, a list space command is executed automatically.

```
COMMAND (?):  LA

VOLUME NAME = VOLCAN
OWNER ID = J SMITH
SYSTEM CODE = IBMEDXS1

DATA SET NAME   TYPE START     SIZE  BOE   EOE

ACCTPAY           H     53       10 01001 01010
ACCTREC           H    163     1500 03007 31124 LAST (02)
CUSTFILE          H   1663      100 31125 33120
BILLING           H    138       23 02108 03004 CONTINUED (01)
```

The listing can be cancelled by the attention command CA.

LI — Redirect Listings to Another Terminal

Use the LI command to direct listings to another terminal. If the terminal from which you loaded $HXUT1 is to be specified, do not enter data.

```
COMMAND (?):  LI

LISTING TERMINAL:  TERM03
```

$HXUT1 - H-Exchange Utility *(continued)*

LS — List Space in H-Exchange Volume

Use the LS command to list the space available for data set definitions and data in the current volume.

```
COMMAND (?):  LS

THE NUMBER OF DEFINED DATA SETS IS  4
THE NUMBER OF UNUSED DATA SET ENTRIES IS 67
THE LARGEST EXTENT AVAILABLE CONTAINS 2138 RECORDS

DISPLAY THE LIST OF AVAILABLE SPACE (Y/N)?  Y

FIRST REC  SIZE


    63   75
   161    2
  1763 2138
```

RE — Rename an H-Exchange Volume

Use the RE command to rename an H-exchange volume.

```
COMMAND (?):  RE

VOLUME NAME (NOW IS 'ABC'):  DEF
RENAME COMPLETED
```

$HXUT1 - H-Exchange Utility *(continued)*

RX — Read an H-Exchange Data Set into an EDX Data Set

Use the RX command to copy an H-exchange data set to an EDX data set. If the EDX target data set already exists, it is not reallocated. If the target EDX data set does not exist, it is automatically allocated. If the H-exchange data set resides on a single diskette, the target size is that of the H-exchange data set. Otherwise, you are asked for the number of diskettes that the multivolume data set spans. For each diskette, 3848 records are allocated in the target data set. If the entire data set is copied, the EDX data set end of data is set to a value corresponding to the H-exchange end of data.

Note: You must enter the H-exchange volume name. There is no default value defined.

Example 1: Copy an entire single volume H-exchange data set to an EDX data set.

```
COMMAND (?):   RX

SOURCE EXCHANGE DATA SET (NAME, VOLUME):   CUSTFILE,CANTST
COPY ENTIRE DATA SET? (Y/N):   Y

TARGET EDX DATA SET (NAME, VOLUME):   CUSTFILE,EDX003

CUSTFILE ALLOCATED SIZE IS 52 RECORDS

COPY OF CUSTFILE,CANTST TO CUSTFILE,EDX003 COMPLETE
TOTAL NUMBER OF RECORDS COPIED WAS 52
```

Example 2: Copy part of a single volume H-exchange data set to an EDX data set.

```
COMMAND (?):   RX

SOURCE EXCHANGE DATA SET (NAME, VOLUME):   CUSTFILE,CANTST
COPY ENTIRE DATA SET? (Y/N):   N
STARTING RECORD NUMBER:  5
COPY TO END OF DATA SET? (Y/N):   Y

TARGET EDX DATA SET (NAME, VOLUME):   TEST,CAN
STARTING RECORD NUMBER:   1

TEST ALLOCATED, SIZE IS 20 RECORDS

COPY OF CUSTFILE,CANTST TO TEST,CAN COMPLETE
TOTAL NUMBER OF RECORDS COPIED WAS 20
```

$HXUT1 - H-Exchange Utility *(continued)*

Example 3: Copy an entire multivolume data set to an EDX data set. This example prompts for the volume name because the H-exchange "Naming Conventions" on page UT-374 were not used.

```
COMMAND (?):  RX

SOURCE EXCHANGE DATA SET (NAME, VOLUME):  $EDXL,CANTST
COPY ENTIRE DATA SET? (Y/N):  Y

TARGET EDX DATA SET (NAME, VOLUME):  $EDXL,CAN

HOW MANY DISKETTES (1-99):  2

$EDXL ALLOCATED, SIZE IS 7696 RECORDS

THIS IS A MULTIVOLUME COPY

SOURCE MAY NOT FIT INTO TARGET

PLEASE MOUNT DISKETTE WITH/FOR
NEXT SEGMENT (02) OF DATA SET $EDXL

VOLUME NAME:  VOLCAN

COPY OF $EDXL,VOLCAN TO $EDXL,CAN COMPLETE
TOTAL NUMBER OF RECORDS COPIED WAS 5803
```

$HXUT1 - H-Exchange Utility *(continued)*

Example 4: Copy an entire multivolume data set to an EDX data set. The multivolume data set spans two diskettes but only contains 161 records. You will need only one diskette to hold the EDX data set because 3848 records are allocated in the target data set, and the source H-exchange data set has less than 3848 records. When the prompt 'HOW MANY DISKETTES (1-99)' is issued, you can enter **1**.

```
COMMAND (?):  RX

SOURCE EXCHANGE DATA SET (NAME, VOLUME):  $EDXL,CANTST
COPY ENTIRE DATA SET? (Y/N):

TARGET EDX DATA SET (NAME, VOLUME):  $EDXL,CAN

HOW MANY DISKETTES (1-99):  1

$EDXL ALLOCATED, SIZE IS 3848 RECORDS

THIS IS A MULTIVOLUME COPY

SOURCE MAY NOT FIT INTO TARGET

PLEASE MOUNT DISKETTE WITH/FOR
NEXT SEGMENT (02) OF DATA SET $EDXL

VOLUME NAME:  VOLCAN

COPY OF $EDXL,VOLCAN TO $EDXL,CAN COMPLETE
TOTAL NUMBER OF RECORDS COPIED WAS 161
```

$HXUT1 - H-Exchange Utility *(continued)*

UD — Update H-Exchange Data Set Label by Name

Use the UD command to update the data set definition or (HDR1) label by entering the data set name associated with that label. If a field is not in conformity with the standard, the utility lets you know about it. In the following example, the end of data extent is changed from 01123 to 40001, and the write protect indicator is set.

```
COMMAND (?):   UD

DATA SET NAME:   ACCTREC

HEADER IDENTIFIER LABEL (NOW IS 'HDR1'):
DATA SET NAME (NOW IS 'ACCTREC  '):
BLOCK LENGTH (NOW IS '  256'):
RECORD ATTRIBUTE (NOW IS ' '):
BEGINNING OF EXTENT (NOW IS '01123'):
PHYSICAL RECORD LENGTH INDICATOR (NOW IS ' '):
END OF EXTENT (NOW IS '40001'):
RECORD/BLOCK FORMAT (NOW IS ' '):
BYPASS INDICATOR (NOW IS ' '):
SECURITY INDICATOR (NOW IS ' '):
WRITE PROTECT INDICATOR (NOW IS ' '):   P
EXCHANGE TYPE (NOW IS 'H'):
MULTIVOLUME INDICATOR (NOW IS 'C'):
VOLUME SEQUENCE NUMBER (NOW IS '02'):
CREATION DATE (NOW IS '      '):
RECORD LENGTH (NOW IS ' '):
OFFSET TO NEXT RECORD SPACE (NOW IS ' '):
EXPIRATION DATE (NOW IS '      '):
VERIFY/COPY INDICATOR (NOW IS ' '):
DATA SET ORGANIZATION (NOW IS ' '):
END OF DATA (NOW IS '01123'):   40001
SYSTEM CODE (NOW IS 'IBMEDXS1'):
FILE ACCESS TYPE (NOW IS ' '):
```

If listing redirection is in affect, using the LI command, the system requests no input.

$HXUT1 - H-Exchange Utility *(continued)*

UH — Update H-Exchange Data Set Label by Number

Use the UH command to update the data set definition or (HDR1) label by entering the positional number for the label. Enter N to refer to the Nth label. If a field is not in conformity with the standard, the utility lets you know about it.

```
COMMAND (?): UH

HDR1 NUMBER: 13

HEADER IDENTIFIER LABEL (NOW IS 'HDR1'):
DATA SET NAME (NOW IS 'ACCTREC'):
BLOCK LENGTH (NOW IS '256'):
RECORD ATTRIBUTE (NOW IS ' '):
BEGINNING OF EXTENT (NOW IS '01123'):
PHYSICAL RECORD LENGTH INDICATOR (NOW IS ' '):
END OF EXTENT (NOW IS '40001'):
RECORD/BLOCK FORMAT (NOW IS ' '):
BYPASS INDICATOR (NOW IS ' '):
SECURITY INDICATOR (NOW IS ' '):
WRITE PROTECT INDICATOR (NOW IS ' '):
EXCHANGE TYPE (NOW IS 'H'):
MULTIVOLUME INDICATOR (NOW IS 'C'):
VOLUME SEQUENCE NUMBER (NOW IS '02'):
CREATION DATE (NOW IS '      '):
RECORD LENGTH (NOW IS ' '):
OFFSET TO NEXT RECORD SPACE (NOW IS ' '):
EXPIRATION DATE (NOW IS '      '):
VERIFY/COPY INDICATOR (NOW IS ' '':
DATA SET ORGANIZATION (NOW IS ' '':
END OF DATA (NOW IS '01123'): 40001
SYSTEM CODE (NOW IS 'IBMEDXS1'):
FILE ACCESS TYPE (NOW IS ' '):
```

UV — Update H-Exchange Volume Label

Use the UV command to update the volume label. If a field is not in conformity with the standard, the utility will let you know about it and force you to reenter it.

```
COMMAND (?): UV

VOLUME LABEL IDENTIFIER (NOW IS 'VOLCAN'):
VOLUME NAME (NOW IS 'VOLCAN'):
ACCESSIBILITY INDICATOR (NOW IS ' '):
SYSTEM CODE (NOW IS 'IBMRPSS1'): IBMEDXS1
OWNER IDENTIFIER (NOW IS ' '): J SMITH
LABEL EXTENSION INDICATOR (NOW IS ' '):
VOLUME SURFACE INDICATOR (NOW IS 'M'):
EXTENT ARRANGEMENT INDICATOR (NOW IS ' '):
SPECIAL REQUIREMENTS INDICATOR (NOW IS ' '):
PHYSICAL RECORD LENGTH (NOW IS '1'):
PHYSICAL RECORD SEQUENCE CODE (NOW IS ' '):
STANDARD VERSION (NOW IS 'W'):
```

$HXUT1 - H-Exchange Utility *(continued)*

WX — Write H-Exchange Data Set From EDX Data Set

Use the WX command to copy an EDX data set to an H-exchange data set. The H-exchange data set is allocated automatically according to the rules described in the explanation of the allocate command (AL). If the source data set will not fit onto the target volume, the system requests new diskette volumes. Partial copies are supported.

Note: You must enter the H-exchange volume name. There is no default value defined.

Example 1: Copy a small EDX data set to a single volume H-exchange data set.

```
COMMAND (?):  WX

SOURCE EDX DATA SET (NAME,VOLUME):  CUSTFILE,EDX003
COPY ENTIRE DATA SET? (Y/N):  Y

TARGET EXCHANGE DATA SET (NAME,VOLUME):  CUSTFILE,CANTST
CUSTFILE ALLOCATED, SIZE IS 1234 RECORDS

COPY OF CUSTFILE,EDX003 TO CUSTFILE,CANTST COMPLETE
TOTAL NUMBER OF RECORDS COPIED WAS 1234
```

Example 2: Copy a partial EDX data set.

```
COMMAND (?):  WX

SOURCE EDX DATA SET (NAME,VOLUME):  CUSTFILE,EDX003
COPY ENTIRE DATA SET? (Y/N):  N
STARTING RECORD NUMBER:  10
COPY TO END OF DATA SET? (Y/N):  Y

TARGET EXCHANGE DATA SET (NAME,VOLUME):  CUSTFILE,CANTST
STARTING RECORD NUMBER:  5
CUSTFILE ALLOCATED, SIZE IS 20 RECORDS

COPY OF CUSTFILE,EDX003 TO CUSTFILE,CANTST COMPLETE
TOTAL NUMBER OF RECORDS COPIED WAS 16
```

$HXUT1 - H-Exchange Utility *(continued)*

Example 3: An EDX data set may be too large to fit on a single data set. If so, the utility prompts you for the name of the volume where the next portion of the data set is to be copied.

```
COMMAND (?):   WX

SOURCE EDX DATA SET (NAME,VOLUME):   $EDXL,ASMLIB
COPY ENTIRE DATA SET? (Y/N):  Y
TARGET EXCHANGE DATA SET (NAME,VOLUME):   $EDXL,CANTST

$EDXL ALLOCATED, SIZE IS 48 RECORDS

THIS IS A MULTIVOLUME COPY

PLEASE MOUNT DISKETTE WITH/FOR
NEXT SEGMENT (02) OF DATASET $EDXL

VOLUME NAME:   VOLCAN

$EDXL ALLOCATED, SIZE IS 113 RECORDS

COPY OF $EDXL,ASMLIB TO $EDXL,VOLCAN COMPLETE
TOTAL NUMBER OF RECORDS COPIED WAS 161
```

$IMAGE - Define Formatted Screen Image

You can use $IMAGE to create formatted screen images for use with terminals that support static screen functions (for example, 4978, 4979, 4980, or 3101 block mode display terminals). The system stores the image (a formatted screen) in a disk or diskette data set for later retrieval by application programs or by this utility for modification.

Data Set Requirements

Before you invoke $IMAGE, use $DISKUT1 to allocate a disk or diskette data set to store the formatted screen image created by $IMAGE. The formatting information and text are stored in a compressed format to conserve space. A stored screen may be any size from one character position up to an entire physical screen; therefore, the amount of space on disk or diskette required to store a given screen image varies. For most 4978 and 4980 static screens, a data set two records in length is adequate. For complex 4978 and 4980 screen formats, a data set 8 records in length is recommended; for complex 3101 screen formats, a data set 15 records in length is recommended.

Considerations for Using $IMAGE

$IMAGE contains a fixed-size buffer that is used in the creation of screens. Protected data fields are placed in the buffer first and then any remaining space is used for unprotected data fields. As a result, the number of unprotected data fields on a screen that can have predefined data is dependent upon the number of protected data fields.

$IMAGE can format 4978 and/or 3101 images, depending on your requirements. The formatted screen subroutines provide support for both 4978 and 3101 displays by selecting the appropriate parameters. With these subroutines, you can use 4978 or 4980 images for the 4978, 4980, or the 3101. However, the 3101 supports fewer unprotected data fields with default data than the 4978 or 4980. Therefore, the migration of a 4978 or 4980 image to the 3101 may result in the truncation of default data within the unprotected data fields. This truncation does not cause the actual unprotected or protected fields to be lost; however, insufficient FTAB space within an application will make it appear to be lost. The message "3101 UNPROTECTED DATA TRUNCATED" is a warning to be aware that the size of your FTAB buffer must be larger than the average size (300 words) used in $IMAGE. $IMAGE expands its own FTAB area whenever you specify a sufficiently large dynamic storage area at load time. If your display is a 3101 image, you select a subroutine that processes 3101 images. For a description of the formatted screen subroutines, $IMDATA, $IMDEFN, $IMOPEN, and $IMPROT, see the *Event Driven Executive Language Programming Guide.*

The system sets end-of-file to indicate if a 3101 datastream is included with the 4978 or 4980 format. Therefore, you should use formats only with application programs or $IMAGE and not other utilities.

Note: If you define screens with EOL/EOF or DEL keys, the system inserts blanks in the 3101, 4978, and 4980 screen images.

$IMAGE - Define Formatted Screen Image *(continued)*

Formatting Screens with $IMAGE

Formatting a screen image on a 4978 or 4980 consists of defining protected and unprotected areas. The 3101, however, uses attribute characters to further define the visual characteristics of protected and unprotected areas. Attribute characters appear as protected blanks on the display screen. Hence, the characters preceding and following an unprotected area (input area) appear as protected blanks. These attribute characters must be taken into account when designing a screen image that will be displayed on the 4978, 4980, and/or the 3101. A description of defining the screen formats for the 4978, 4980, and the 3101 terminals follows.

For an aid in laying out the format of the screen to be defined, refer to the *IBM 3270 Information Display System Layout Sheet*, GX27-2951.

4978/4980 Screen Format

Visualize a screen format for a 4978 and 4980 as a matrix (usually 24 x 80) of characters that directly represents the image displayed on the screen. The system displays a character at position (m,n) in this matrix at position (m,n) on the screen. A character can possess only one attribute in a 4978 and 4980 image: its protection status (protected or unprotected). As a result, you do not need to specify explicit attributes for characters. Character positions on a 4978 or 4980 screen are defined as follows:

- Protected — positions on the screen where data values are entered during the protected field definition mode (PF1). Once the screen is formatted, these positions will remain as defined until you enter PF1 mode again and change them.

- Unprotected — positions defined with a specified null character during the protected field definition mode (PF1) and referred to as the input fields (areas on the screen that accept operator input). After you have defined the unprotected fields in PF1 mode, you can modify these positions to contain data values by entering the unprotected field definition mode (PF2).

Characters in protected positions are displayed with low intensity; those in unprotected positions are displayed with high intensity.

For an example of formatting a screen image on a 4978 or 4980, refer to the program preparation example in the *Event Driven Executive Language Programming Guide.*

$IMAGE - Define Formatted Screen Image *(continued)*

3101 Screen Format

A screen format designed for a 3101 can also be visualized as a matrix (usually 24 x 80) that directly represents the image previously defined on the display. Additionally, the format consists of attribute characters for use only on a 3101 display. Fields on the display are defined as either protected or unprotected (specified by the defined null character). The attribute characters define the visual attributes of the characters that will follow on the display. Since a number of possible attributes exist, you can format portions of the display into fields by assigning the following attributes.

- High intensity — the characters in a designated field are displayed in a higher intensity than the rest of the display
- Low intensity — the characters in a designated field are displayed in normal intensity
- Nondisplay — the characters in a designated field are not displayed
- Blinking — the characters in a designated field blink.

Note: If you do not use attribute characters, then protected positions are displayed in low intensity; unprotected positions are displayed in high intensity. However, if the first position on the 3101 display is unprotected, the entire first field will be displayed in low intensity. On the 3101, you can define these areas with other attributes, such as blinking versus nonblinking or display versus nondisplay.

Once you have created a screen image using the attribute bytes, each attribute byte occupies a character position on the display and is visually represented by a blank. As a result, the characters preceding and following unprotected fields always appear as protected blanks on a 3101 display. This is because attributes are defaulted, if not explicitly specified, between the protected and unprotected fields on a 3101 display.

$IMAGE - Define Formatted Screen Image *(continued)*

Displaying Screens

When you display a screen designed for a 4978 or 4980 on a 3101, the character preceding and following each unprotected field becomes a blank. This may necessitate slight modifications to such screens. For example, if you have not allowed for proper spacing, you can lose a portion of a protected field. The following areas on a 4978 or 4980 display

```
     NAME:@@@@@<--(5 characters)
```

will be displayed on a 3101 as follows:

```
     NAME @@@@@ --(5 characters)
```

On the 3101 display, the last character (:) in the first protected field and the first character (<) in the second protected field are lost because the characters preceding and following the unprotected field appear as protected blanks and overlay the data values defined in those protected positions.

Screen formats designed for a 3101 that have attribute characters defined can be displayed on a 4978 or 4980. The attribute characters (protected blanks) are ignored and the fields default to the protected and unprotected areas as defined.

Invoking $IMAGE

You invoke $IMAGE with the $L operator command or option 4.4 of the session manager.

```
> $L $IMAGE
LOADING $IMAGE    78P,00:44:30, LP= 9200, PART=1

COMMAND (?):
```

$IMAGE - Define Formatted Screen Image *(continued)*

$IMAGE Commands

When you load $IMAGE, it is in command mode and you can choose any of the available commands. To display the $IMAGE commands at your terminal, enter a question mark in reply to the prompting message COMMAND (?):.

```
COMMAND(?):  ?

ATTR -- DEFINE ATTRIBUTE CHARACTERS
DIMS -- DEFINE SCREEN DIMENSIONS
EDIT -- EDIT SCREEN FORMAT
FTAB -- DISPLAY FIELD TABLE
HTAB -- SET (HORIZONTAL) TABS FOR EDIT
KEYS -- DISPLAY USE OF PF KEYS
NULL -- DEFINE NULL CHARACTER
PRNT -- PRINT SCREEN IMAGE AND TABLES
SAVE -- SAVE SCREEN FORMAT(S) IN DATA SET
VTAB -- SET (VERTICAL) TABS FOR EDIT
END  -- END PROGRAM

COMMAND(?):
```

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

Notes:

1. You can enter these commands in command mode only. They are not available in edit mode. Therefore, you should define the null character, the dimensions, and the tab settings before you enter edit mode (EDIT).

2. The values you enter for the ATTR, DIMS, HTAB, NULL, and VTAB command remain in effect until you change them or you end the utility.

3. If the message "INTERNAL 3101 DATA STREAM TOO LARGE" appears when you enter the EDIT command, then you must increase the size of the buffer containing the 3101 data stream. This can be done only when you load $IMAGE by specifying a sufficiently large value for the storage parameter on the $L command. (Example - $L $IMAGE,,6000.) If you do not specify this value, it defaults to 4608.

$IMAGE - Define Formatted Screen Image *(continued)*

ATTR — Define Attribute Characters

Use the ATTR command to define the characters that represent the attribute bytes on a 3101 display screen. You are prompted with the type of attribute and its existing value. If you do not enter a new character to represent the attribute byte, it defaults to the character displayed. If you wish to delete an attribute character, enter a space to represent the attribute byte.

Note: You must not use an attribute character as a character on the protected portion of the screen. For example:

```
COMMAND (?):  ATTR

ATTRIBUTE CHARACTERS WITH MDT OFF:

  LOW INTENSITY  (NOW IS ' ') =  %

  HIGH INTENSITY (NOW IS ' ') =  *

  BLINKING       (NOW IS ' ') =  $

  NONDISPLAY     (NOW IS ' ') =  &

DO YOU WISH TO DEFINE ATTRIBUTE CHARACTERS WITH MDT ON?  N

COMMAND (?):
```

Once you define attribute bytes, you can change them by issuing the ATTR command again. The ATTR command will display the attribute bytes that are currently defined in the (NOW IS ' ') portion of the prompt and you can change or delete the attribute characters that you no longer need.

If you define an attribute character that is already in use, the following warning message is issued.

```
CHARACTER ALREADY IN USE (WARNING)
```

If you replace a previously defined attributed character with a blank, the following message is issued:

```
CHARACTER REMOVED
```

If you do not enter a character in response to the prompt, the following message is issued:

```
CHARACTER NOT CHANGED
```

$IMAGE - Define Formatted Screen Image *(continued)*

$IMAGE prompts you as to whether or not you want to define characters with the modified data tag (MDT) on. If you respond Y, the fields specified with that particular attribute character will be flagged as having been modified by the operator you can access them with the READTEXT instruction.

DIMS — Define Screen Dimensions

Use the DIMS command, followed by the number of lines and the line size, to define the dimensions of the new screen image you are creating. For example:

```
DIMS  10 20      (10 20-character lines)
DIMS  24 80      (full screen image)
```

Defining the dimensions for a new screen image will destroy any previously defined screen image that has not been saved.

Note: To define a screen image format for the 3101 so that the ATTR command may be used to specify attribute bytes, the dimensions of the screen must be 24 x 80.

EDIT — Enter Screen Mode

Use the EDIT command to place $IMAGE in edit verification mode. If you are going to edit an existing screen image, enter the data set name and volume containing the image with the EDIT command. If you are creating a new screen image, enter EDIT without reference to a data set.

Notes:

1. If you are modifying an existing screen image, you must redefine the null character each time you invoke $IMAGE..

2. The volume, if unspecified, defaults to the IPL volume. If you do not specify a data set name, then the program displays the currently defined image.

Examples

```
EDIT
EDIT  IMAGE1
EDIT  IMAGEB,EDX002
```

$IMAGE - Define Formatted Screen Image *(continued)*

END — End $IMAGE

Use the END (or EN) command to end the $IMAGE utility. If the screen image has been modified since the last SAVE operation, the following question is issued:

```
SAVE FINAL IMAGE?
```

Respond Y to save the image, N if you don't want it saved.

FTAB — Display Field Table

Use the FTAB command to display or print a table, created by $IMAGE, containing the field, row, column, size, and attributes of the unprotected fields in the last screen image created or edited. The table is printed on the device labeled $SYSPRTR unless you specify another device. For example:

```
FTAB           (prints on $SYSPRTR)

FTAB *         (displays on terminal that issued
               the command)

FTAB $SYSLOGA  (prints on device labeled as
               the alternate logging device)
```

If the device you specify is not defined, the table is displayed on the terminal that issued the FTAB command.

Following is an example of a table displayed by FTAB where the character @ is defined as an attribute character by the ATTR command:

```
FIELD  ROW  COLUMN  LENGTH  ATTRIBUTE
  1     7     46      15        @
  2    10     46      15        @
  3    13     46       3        @
  4    16     46       8        @
  5    20     49       1        @
```

Only a screen image formatted with dimensions equal to 24 x 80 will contain the attribute characters defined for each field.

$IMAGE - Define Formatted Screen Image *(continued)*

HTAB — Set (Horizontal) Tabs for Edit

Use the HTAB command to define the horizontal tab settings to be in effect during edit mode. For example:

```
HTAB  10 15 20 40 45 50 73
```

If you do not define HTAB, it defaults to the following settings: 10, 20, 30, 40, 50, 60, and 70. The tab settings allow you to position the cursor to the corresponding display positions with the PF1 key in edit mode. If a tab value exceeds the line size or is not in ascending order, then the cursor moves to the next line when the invalid setting is encountered.

KEYS — Display Use of PF Keys

Use the KEYS command to list the functions of the PF1,PF2, and PF3 keys immediately after you enter the edit mode.

```
COMMAND(?):  KEYS

PF1 - MODIFY PROTECTED FIELDS
PF2 - MODIFY DATA FIELDS
PF3 - ENTER COMMAND MODE

COMMAND(?):
```

Once you have chosen either PF1 or PF2 to enter data, PF1 and PF2 function as the horizontal and vertical tab keys (respectively) when you are defining protected or data fields in EDIT mode.

$IMAGE - Define Formatted Screen Image *(continued)*

NULL — Define Null Character

Use the NULL command to define a character that is interpreted as the null character during editing of the image. In edit mode, you enter a null character in each position in which you want to display unprotected data or which is to accept data entered by the operator. For example:

```
COMMAND (?):  NULL .
COMMAND (?):  NULL ,
COMMAND (?):  NULL
```

Once you have defined the null character, you can change it by issuing the NULL command again. You are prompted with the existing value for the null character. The NULL command displays the null character currently defined in the (NOW IS ' ') portion of the prompt. If you do not enter a new character to represent the null character, it defaults to the character displayed. For example:

```
COMMAND (?):  NULL
NULL         (NOW IS ',') =
```

If you define a null character that is already in use, the following warning message is issued:

```
NULL CHARACTER ALREADY IN USE (WARNING)
```

If you replace a previously defined null character with a blank, the following message is issued:

```
CHARACTER REMOVED
```

If you do not enter a character in response to the prompt, the following message is issued:

```
NULL CHARACTER NOT CHANGED
```

Notes:

1. If you are modifying an existing screen image, you must redefine the null character each time you invoke $IMAGE.

2. If the NULL command is not defined prior to an editing session, the null character defaults to a period (.). As a result, all periods defined as protected assume an unprotected status in subsequently edited screens.

$IMAGE - Define Formatted Screen Image *(continued)*

PRNT — Print Screen Images and Tables

Use the PRNT command to display or print the screen image of the last screen created or edited as it appears when in PF1 mode (protected field definition mode). The screen image is printed on the device labeled $SYSPRTR unless you specify another device. For example:

```
PRNT           (prints on $SYSPRTR)

PRNT *         (displays on terminal that issued
               the command)

PRNT $SYSLOGA  (prints on device labeled as
               the alternate logging device)
```

If the device you specify is not defined, the screen image is displayed on the terminal that issued the PRNT command.

The screen image displayed by PRNT is surrounded by a box numbered in increments of five to allow column verification. The defined null character is displayed and, if the dimensions of the screen equal 24 x 80, the defined attribute characters are also displayed.

Sample PRNT displays: **The dimensions of the following screen image do not equal 24 x 80; therefore, defined attribute characters are not displayed.**

```
DIMS   12  40

   0    5    10   15   20   25   30   35
   +----+----+----+----+----+----+----+----
 0+$IMAGE SUBSCREEN                        +0
  |                                        |
  |UNPROTECTED FIELDS WITHOUT DEFAULT DATA |
  |FIELD1 .... FIELD2 ........ FIELD3 .    |
  |                                        |
 5+UNPROTECTED FIELDS WITH DEFAULT DATA:   +5
  |FIELD4 .... FIELD5 ........ FIELD6 .    |
  |                                        |
  |                                        |
  |                                        |
10+                                        +10
  |THIS SCREEN IS 12 BY 20 IN SIZE         |
   +----+----+----+----+----+----+----+----+
   0    5    10   15   20   25   30   35

NULL CHARACTER IS '.'
```

The dimensions of the following screen image equal 24 x 80; therefore, PRNT displays the defined attribute characters used in formatting the screen image.

$IMAGE - Define Formatted Screen Image *(continued)*

```
DIMS:   24     80

   0    5    10   15   20   25   30   35   40   45   50   55   60   65   70   75
   +----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----
 0+$SMM0209: SESSION MANAGER $UPDATE PARAMETER INPUT MENU--------------------------+0
  |ENTER/SELECT PARAMETERS:                                    DEPRESS PF3 to RETURN|
  |                                                                                 |
  |                                                                                 |
  |                                                                                 |
 5+                                                                                 +5
  |                                                                                 |
  |    OBJECT INPUT   (NAME,VOLUME) ===========>  ...............                   |
  |                                                                                 |
  |                                                                                 |
10+    PROGRAM OUTPUT (NAME,VOLUME) ===========>  ...............                   +10
  |                                                                                 |
  |                                                                                 |
  |    REPLACE (ENTER YES IF PROGRAM EXISTS) ==>  ...                               |
  |                                                                                 |
15+                                                                                 +15
  |    LISTING (TERMINAL NAME / *) ============>  ........                          |
  |                                                                                 |
  |                                                                                 |
  |    NOTE: THE OBJECT INPUT< PROGRAM OUTPUT AND LISTING TERMINAL NAME ARE         |
20+          REQUIRED PARAMETERS AND MUST BE ENTERED.                               +20
  |          AN '*' MAY BE USED TO SEPCIFY THIS TERMINAL AS THE LISTING TERMINAL    |
  |                                                                                 |
  |                                                                                 |
   +----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----
   0    5    10   15   20   25   30   35   40   45   50   55   60   65   70   75
NULL CHARACTER IS '.'

ATTRIBUTE CHARACTERS WITH MDT OFF:
  LOW INTENSITY IS  ' '
  HIGH INTENSITY IS ' '
  BLINKING IS       ' '
  NONDISPLAY IS     ' '

ATTRIBUTE CHARACTERS WITH MDT ON:
  LOW INTENSITY IS  ' '
  HIGH INTENSITY IS ' '
  BLINKING IS       ' '
  NONDISPLAY IS     ' '
```

SAVE — Save Screen Format(s) in Data Set

Use the SAVE command to save the screen image format. If you do not specify a data set name with the SAVE command, you are issued a prompt asking you if you wish to save the format on the previously specified data set. For example, if you edited a data set BILLSCRN,SCRNS, the following occurs.

```
COMMAND (?):  SAVE

SHOULD THE 3101 DATASTREAM BE SAVED?  N

SAVE FORMAT ON BILLSCRN,SCRNS?  Y
```

$IMAGE - Define Formatted Screen Image *(continued)*

If you respond with an N, you are prompted for a new format data set name as follows:

```
ENTER FORMAT DATA SET NAME:
```

The formatting information and text that define the image are stored in the data set in a special packed format. For packed format information, refer to the *Event Driven Executive Language Programming Guide*. Some examples are:

```
SAVE
SAVE  IMAGE2
SAVE  IMAGE2,EDX003
```

If the dimensions of the screen image are 24 x 80, you are prompted as follows:

```
SHOULD THE 3101 DATASTREAM BE SAVED?
```

If you respond Y, the image will be saved in the same data set as the 4978 or 4980 screen image and later retrieved for display on the 3101. You can also perform this operation by using the DSOPEN subroutine or defining the data set at program load time and issuing the disk READ instruction. For more information on retrieving screen image format, see the $IMOPEN subroutine in the *Event Driven Executive Language Programming Guide.*

When the image has been saved, the following messages appear, with the number of 256-byte records indicated in parentheses.

```
THE FORMAT(S) WILL BE SAVED ON data set
SAVED  (n RECORDS)

4978 FORMAT REQUIRES xxxx BYTES
3101 FORMAT REQUIRES yyyy BYTES
```

Note: If the data set you specified does not exist, it is allocated automatically. If the specified data set exists but is too small to contain the format(s), it is deleted and a new one allocated.

$IMAGE - Define Formatted Screen Image *(continued)*

VTAB — Set (Vertical) Tabs for Edit

Use the VTAB command to define vertical tab settings to edit the screen image conveniently by columns rather than rows. The default vertical tabs range from 1 to 24 in 1-line increments. You can redefine these as in the following example:

```
VTAB      5 10 20 24
```

In edit mode, when you press the PF2 key (vertical tab key), the cursor moves to the next line indicated at the last encountered horizontal tab setting. When the last vertical tab setting is passed, the cursor moves to line 0 at the next horizontal tab setting.

Entering Edit Verification Mode

Once you have defined the null character, the dimensions, and the tab settings, the last command you enter is EDIT. EDIT places $IMAGE into edit verification mode. If you will be editing an existing screen image format, the dimensions used will be those of the save screen image format. Otherwise, EDIT is entered without reference to a data set. Edit verification mode displays the screen image as you would see it using the $IMOPEN, $IMDEFN, $IMPROT, and $IMDATA subroutines. See the *Event Driven Executive Language Programming Guide* for information on using these subroutines.

When you enter edit verification mode, the PF keys have the following functions:

- PF1 — define protected fields
- PF2 — define data fields (unprotected)
- PF3 — return to command mode

However, as soon as you press either PF1 or PF2 entering edit verification mode (indicating definition of protected or unprotected fields), the function of PF1 and PF2 is redefined. PF1 is then used as the horizontal tab key and PF2 as the vertical tab key.

To exit protected or unprotected mode and return to edit verification mode, press the ENTER key.

The following figure shows the functions available under $IMAGE. However, its primary purpose is to show the sequence of events that occurs when $IMAGE enters edit mode.

$IMAGE - Define Formatted Screen Image *(continued)*

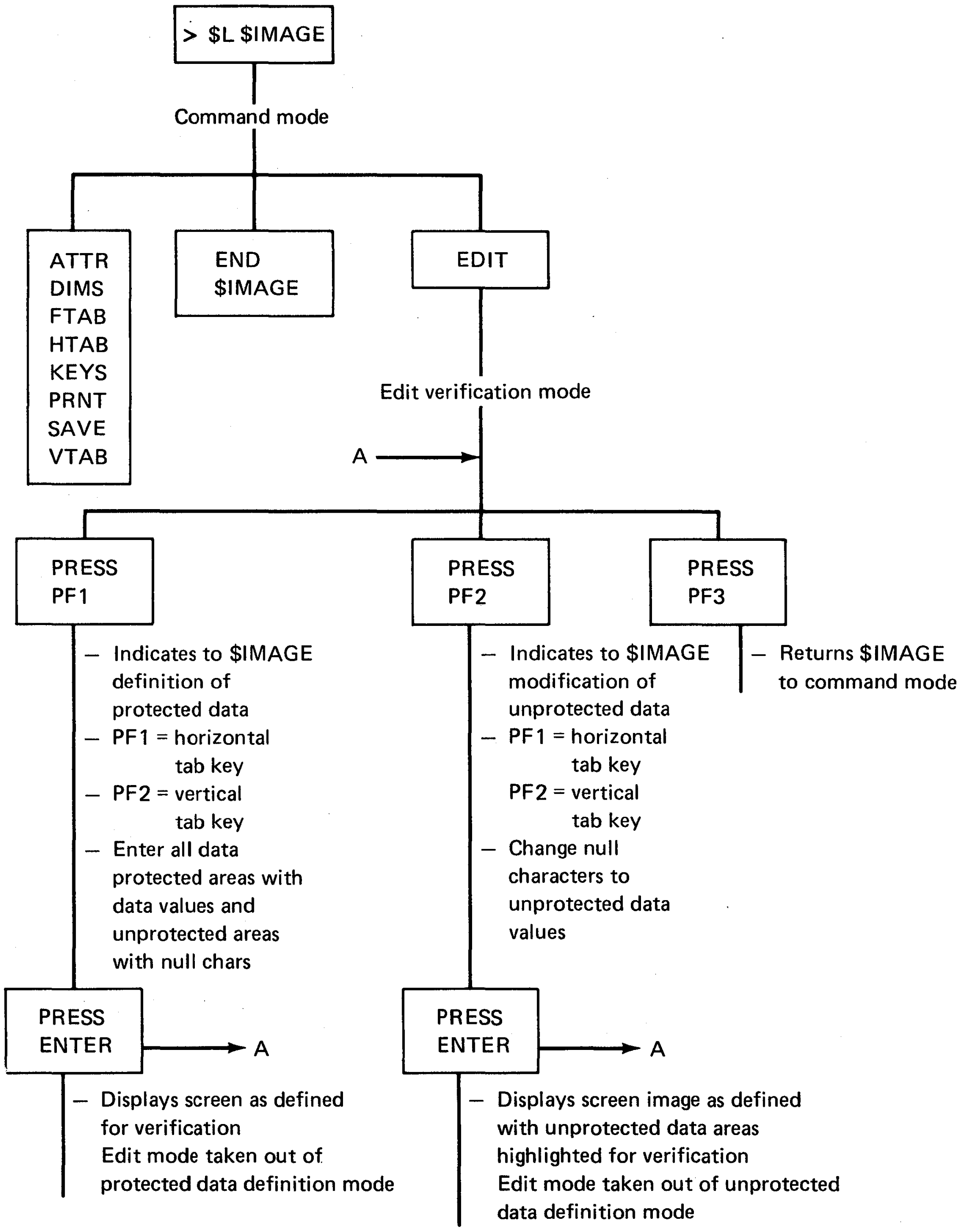

> $L $IMAGE
Command mode
ATTR
DIMS
FTAB
HTAB
KEYS
PRNT
SAVE
VTAB
END $IMAGE
EDIT
Edit verification mode
A
PRESS PF1
– Indicates to $IMAGE definition of protected data
– PF1 = horizontal tab key
– PF2 = vertical tab key
– Enter all data protected areas with data values and unprotected areas with null chars
PRESS ENTER
A
– Displays screen as defined for verification Edit mode taken out of protected data definition mode
PRESS PF2
– Indicates to $IMAGE modification of unprotected data
– PF1 = horizontal tab key PF2 = vertical tab key
– Change null characters to unprotected data values
PRESS ENTER
A
– Displays screen image as defined with unprotected data areas highlighted for verification Edit mode taken out of unprotected data definition mode
PRESS PF3
– Returns $IMAGE to command mode

Figure 21. $IMAGE - Command and Edit Mode

$IMAGE - Define Formatted Screen Image *(continued)*

Defining Protected Fields

When protected and unprotected text is to appear on a screen created by $IMAGE, enter the protected data first. Press PF1 to signal the utility that protected fields are to be defined.

When you define the protected areas of a screen image, all characters you enter, other than the defined null character, are protected data. All areas of the screen that don't contain the null character will be protected when the screen is completed. The unprotected areas of the screen (indicated with the defined null character) should then be specified. You can modify these unprotected areas later to contain contain data values.

Once all areas (protected and unprotected) on the screen have been defined, press the ENTER key to take the utility out of protected field definition mode. The screen as defined is displayed so you can verify if it is correct. PF1 and PF2 again have the meanings printed out by the KEYS command and no longer function as tab keys.

Entering Unprotected Output Fields

To modify unprotected areas on the image, press PF2. Pressing PF2 allows you to modify the unprotected fields and to display the screen showing both the protected areas and the null characters representing the unprotected areas. Now you can fill in the unprotected fields with data values. You can leave other null fields alone since they will be used as input fields to provide information to your application program.

After you have modified the unprotected fields, press the ENTER key to take the utility out of unprotected field definition mode. The screen as defined is displayed so you can verify if it is correct. PF1 and PF2 again have the meanings printed out by the KEYS command and no longer function as tab keys. If you want to make any changes to the screen, press PF1 to allow protected field entry or PF2 to allow unprotected field entry.

Saving the Image Created

If the image is correct, press PF3 to return to command mode. The screen is blanked and you are prompted for a command. Enter the SAVE command, followed by the name of the data set that you allocated for this purpose. The image will be saved and you can end the $IMAGE utility with the END command.

$INITDSK - Initialize Direct Access Device; Volume Control

$INITDSK performs initialization operations for direct access storage devices.

$INITDSK performs the following functions:

- Initialize a device by writing:
  - the volume directory on a disk device or a multivolume diskette, with the option to specifying the record number at which the system will write the directory;
  - owner identification on a diskette;
  - the volume label on a single-volume diskette;
  - a data set directory in a disk or diskette volume;
  - IPL text on a disk or diskette device.
- Allocate a:
  - volume on a disk device or multiple-volume diskette;
  - volume under the fixed heads of a disk device.
- Delete a volume on a disk device or multiple volume diskette.
- List the volume(s) on a disk or diskette device.
- Split a disk or multiple volume diskette volume into two volumes.
- Verify a volume by reading all or part of a disk or diskette volume and listing any I/O errors.
- Verify a device by reading all or part of a disk or diskette and listing any I/O errors.
- Rename a volume on a disk or multivolume diskette device.
- Rename a diskette VOL1 label and owner id.

Invoking $INITDSK

You invoke $INITDSK with the $L operator command, option 3.7 of the session manager, or through the $DASDI utility. During diskette initialization for Event Driven Executive format, $DASDI gives you the option of going directly into $INITDSK operation without having to end $DASDI and issue the $L command for $INITDSK. You must vary diskette devices online before you invoke $INITDSK.

$INITDSK - Initialize Direct Access Device Volume Control *(continued)*

$INITDSK Commands

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

```
COMMAND (?): ?

ID - INITIALIZE DEVICE
AV - ALLOCATE VOLUME
AF - ALLOCATE FIXED HEAD VOLUME
SV - SPLIT VOLUME
IV - INITIALIZE VOLUME
II - INITIALIZE IPL TEXT
VD - VERIFY DEVICE
VV - VERIFY VOLUME
LP - LIST PERFORMANCE VOLUMES
LV - LIST VOLUMES... LV DEVADDR PRTNAME
LAV- LIST ALL VOLUMES... LAV PRTNAME
DV - DELETE VOLUMES
RV - RENAME VOLUMES
RD - RENAME DISKETTE VOL1/OWNERID
WV - SET WRITE VERIFY
WN - CLEAR WRITE VERIFY
EN - END PROGRAM

COMMAND (?):
```

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

Note: When you specify a volume for the SV, IV, and VV commands, you must make the volume name unique. If $INITDSK finds duplicate names, it uses the first one it finds.

Each command and its explanation appears in alphabetical order on the following pages.

$INITDSK - Initialize Direct Access Device Volume Control *(continued)*

AF — Allocate Fixed-Head Volume

Use the AF command to allocate the fixed-head area of a disk as a separate volume. Enter information about the volume in the volume directory for the device.

You cannot use AF for diskette devices.

Example: Allocate a fixed-head volume.

```
COMMAND (?): AF
DEVICE ADDRESS: 48
VOLUME NAME: FHVOL
FHVOL   ALLOCATED

COMMAND (?):
```

AV — Allocate Volume

Use the AV command to allocate a volume on a disk or multivolume diskette device. AV allocates one volume at a time; for multiple volumes, you must use AV repeatedly for each volume you allocate. $INITDSK prompts you for the following information:

- The address (in hexadecimal) of the device you want allocated.
- The name (1-6 characters) of the volume you want allocated.
- The number (in decimal) of the records within the volume.

A volume can contain from three records to the maximum number of records on a single device. Enter information about the volume in the volume directory for the device. In addition, AV gives you the option of initializing (writing) a data set directory for the volume you are allocating. You must have a data set directory before you can allocate individual data sets.

Once you allocate a volume, you may want to:

1. Initialize (IV) the volume just allocated.
2. Set (WV) the write verify function on.
3. Copy the current system nucleus to the volume just allocated.
4. Write (II) IPL text to point to the system nucleus.

The AV command prompts you for these steps. If you do not want to perform any of these functions, respond N to step 1. You can also perform these functions by using the IV, WV, and II commands individually. For an explanation of each step, see the individual command.

$INITDSK - Initialize Direct Access Device Volume Control *(continued)*

Example 1: Allocate and initialize a disk volume.

```
COMMAND (?): AV
DEVICE ADDRESS: 48
VOLUME: EDX002
SIZE IN RECORDS: 50000
EDX002   ALLOCATED
INITIALIZE THE VOLUME JUST ALLOCATED? Y
MAXIMUM NUMBER OF DATA SETS: 10
DO YOU WANT WRITE VERIFY FOR THIS VOLUME? N
ALLOCATE $EDXNUC? N
VOLUME INITIALIZED

COMMAND (?):
```

Example 2: Allocate and initialize a disk volume, set write verify on, copy the system nucleus, and write the IPL text to point to the system nucleus copied to the volume.

```
COMMAND (?): AV
DEVICE ADDRESS: 48
VOLUME: EDX002
SIZE IN RECORDS: 50000
EDX002   ALLOCATED
INITIALIZE THE VOLUME JUST ALLOCATED? Y
MAXIMUM NUMBER OF DATA SETS: 10
DO YOU WANT WRITE VERIFY FOR THIS VOLUME? Y
ALLOCATE $EDXNUC? Y
VOLUME INITIALIZED
INITIALIZE IPL TEXT? Y
IPL TEXT WRITTEN

COMMAND (?):
```

Example 3: Allocate and initialize a diskette volume.

```
COMMAND (?): AV
DEVICE ADDRESS: 02
VOLUME: MYVOL
SIZE IN RECORDS: 5000
MYVOL   ALLOCATED
INITIALIZE THE VOLUME JUST ALLOCATED? Y
MAXIMUM NUMBER OF DATA SETS: 10
DO YOU WANT WRITE VERIFY FOR THIS VOLUME? N
VOLUME INITIALIZED

COMMAND (?):
```

$INITDSK - Initialize Direct Access Device Volume Control *(continued)*

DV — Delete Volumes

Use the DV command to delete a volume. Then you can reallocate the volume with the AV or AF commands. When you delete a volume, you can no longer access the data within that volume. $INITDSK prompts you for the address and name of the volume you want deleted.

Example: Delete a volume.

```
COMMAND (?): DV
ADDRESS: 03
VOLUME: EDX002
EDX002 DELETE? Y
EDX002 DELETED
```

EN — End $INITDSK

Use the EN command to end the $INITDSK utility.

Example: End $INITDSK.

```
COMMAND: EN
$INITDSK ENDED AT 00:06:59
```

$INITDSK - Initialize Direct Access Device Volume Control *(continued)*

ID — Initialize Device

Use the ID command to write the volume directory on a 4962, 4963, 4967, DDSK-30 ,DDSK-60 disk device, or a multiple volume diskette.

Note: Be sure to format a new diskette with primary option 1 of the $DASDI utility before initializing it with the ID command.

You must initialize each disk device with a volume directory which INITDSK writes at a default record number. When you use ID, you may write the directory number to a record number other than the default. $INITDSK prompts you for the size of the device, the directory default, and its current record number. Once you use ID, you can no longer access the data that the disk device or multiple volume diskette contained previously.

ID also writes a volume label and owner identifier on a diskette. The volume label must be 1 to 6 characters; the identification must be 1 to 14 characters.

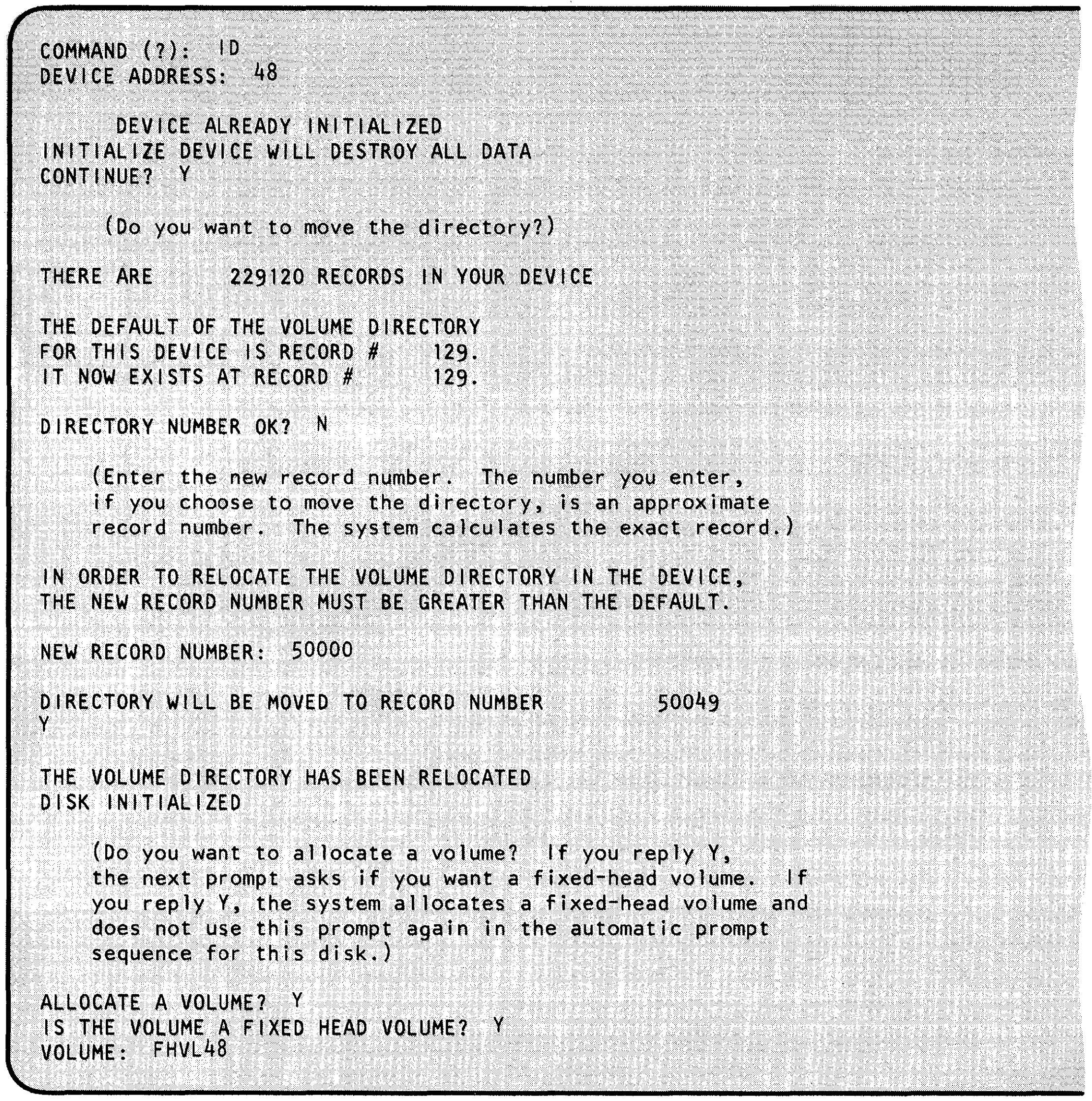
Once you initialize a disk or diskette, you may want to:

1. Allocate a volume(s) or a fixed-head volume on the disk.
2. Initialize the disk or diskette volume(s).
3. Set write verify on.
4. Copy the current system nucleus.
5. Write IPL text on a disk or diskette volume.

The ID command prompts you for all of the above steps for disk or multiple volume diskette and steps 2 and 5 for diskette to perform these functions. You can also perform the functions by invoking the AF, AV, II, IV, or WV commands individually.

$INITDSK - Initialize Direct Access Device Volume Control *(continued)*

Example 1: Initialize a disk volume. $INITDSK prompts you for the information required to initialize a disk volume.



```
COMMAND (?):  ID
DEVICE ADDRESS:  48

      DEVICE ALREADY INITIALIZED
INITIALIZE DEVICE WILL DESTROY ALL DATA
CONTINUE?  Y

     (Do you want to move the directory?)

THERE ARE      229120 RECORDS IN YOUR DEVICE

THE DEFAULT OF THE VOLUME DIRECTORY
FOR THIS DEVICE IS RECORD #    129.
IT NOW EXISTS AT RECORD #      129.

DIRECTORY NUMBER OK?  N

    (Enter the new record number.  The number you enter,
    if you choose to move the directory, is an approximate
    record number.  The system calculates the exact record.)

IN ORDER TO RELOCATE THE VOLUME DIRECTORY IN THE DEVICE,
THE NEW RECORD NUMBER MUST BE GREATER THAN THE DEFAULT.

NEW RECORD NUMBER:  50000

DIRECTORY WILL BE MOVED TO RECORD NUMBER         50049
Y

THE VOLUME DIRECTORY HAS BEEN RELOCATED
DISK INITIALIZED

    (Do you want to allocate a volume?  If you reply Y,
    the next prompt asks if you want a fixed-head volume.  If
    you reply Y, the system allocates a fixed-head volume and
    does not use this prompt again in the automatic prompt
    sequence for this disk.)

ALLOCATE A VOLUME?  Y
IS THE VOLUME A FIXED HEAD VOLUME?  Y
VOLUME:  FHVL48
```

Figure 22 (Part 1 of 2). Initialize a disk

$INITDSK - Initialize Direct Access Device Volume Control *(continued)*

```
    (Do you want to initialize the volume?)

INITIALIZE THE VOLUME JUST ALLOCATED?  Y
MAXIMUM NUMBER OF DATASETS:  50

    (Do you want to set write verify on?)

DO YOU WANT TO WRITE VERIFY FOR THIS VOLUME?  N

    (Do you want to allocate space for a system nucleus
    on this volume?)

ALLOCATE $EDXNUC?  N
VOLUME INITIALIZED

   (The system repeats the prompt sequence for allocating
   a volume, initializing that volume, and writing IPL text
   on the volume.)

ALLOCATE ANOTHER VOLUME?  Y
VOLUME:  VOL1
SIZE IN RECORDS:  1000
VOL1     ALLOCATED
INITIALIZE THE VOLUME JUST ALLOCATED?  Y
MAXIMUM NUMBER OF DATA SETS:  100
DO YOU WANT WRITE VERIFY FOR THIS VOLUME?  N
ALLOCATE $EDXNUC?  N
VOLUME INITIALIZED
ALLOCATE ANOTHER VOLUME?  Y
VOLUME:  VOL2
SIZE IN RECORDS:  20000
VOL2     ALLOCATED
INITIALIZE THE VOLUME JUST ALLOCATED?  Y
MAXIMUM NUMBER OF DATA SETS:  100
DO YOU WANT WRITE VERIFY FOR THIS VOLUME? N
ALLOCATE $EDXNUC?  Y
VOLUME INITIALIZED
INITIALIZE IPL TEXT?  Y
IPL TEXT WRITTEN

   (The system repeats the prompt sequence until
   you respond N to the allocate-volume prompt.)

ALLOCATE ANOTHER VOLUME?  N
COMMAND (?):  EN
$INITDSK ENDED AT 01:00:47
```

Figure 22 (Part 2 of 2). Initialize a disk

$INITDSK - Initialize Direct Access Device Volume Control *(continued)*

Example 2: Initialize a single-volume diskette. $INITDSK prompts you for volume and IPL text initialization. If you initialize the diskette as a multivolume diskette, the system issues a series of automatic prompts until you respond N to the allocate-volume prompt. If you do not initialize the diskette as a multivolume diskette, the series of automatic prompts occurs only once for the diskette you are initializing.

```
COMMAND (?): ID
DEVICE ADDRESS: 22
SLOT NUMBER: 2

      DEVICE ALREADY INITIALIZED
INITIALIZE DEVICE WILL DESTROY ALL DATA
CONTINUE? Y
INITIALIZE DISKETTE AS MULTIVOLUME TYPE? N
NEW VOLUME LABEL: EDX001
ENTER OWNER IDENTIFICATION: DEPT A10
SINGLE VOLUME TYPE DISKETTE INITIALIZED

   (Do you want to initialize the volume?)

INITIALIZE THE VOLUME? Y
VOLUME ALREADY INITIALIZED; CONTINUE? Y
MAXIMUM NUMBER OF DATA SETS: 200

   (Do you want to set write verify on?)

DO YOU WANT WRITE VERIFY FOR THIS VOLUME? N

   (Do you want to allocate space for a system nucleus
   on this volume?)

ALLOCATE $EDXNUC? Y
VOLUME INITIALIZED

   (Do you want to write IPL text on the volume?)

INITIALIZE IPL TEXT? Y
IPL TEXT WRITTEN

COMMAND (?):
```

Figure 23. Initialize a diskette

$INITDSK - Initialize Direct Access Device Volume Control *(continued)*

Example 3: Initialize a multivolume diskette.

```
COMMAND (?): ID
DEVICE ADDRESS: 02
INITIALIZE DEVICE MAY DESTROY ALL DATA
CONTINUE? Y
INITIALIZE DISKETTE AS MULTIVOLUME TYPE? Y
NEW VOLUME LABEL: EDX005
ENTER OWNER IDENTIFICATION: DEPT A10
MULTIVOLUME TYPE DISKETTE INITIALIZED

   ($INITDSK repeats the prompts for a
   multivolume diskette until you respond
   N to the ALLOCATE A VOLUME? prompt.)

ALLOCATE A VOLUME? Y
VOLUME: VOL1
SIZE IN RECORDS: 100
VOL1      ALLOCATED
INITIALIZE THE VOLUME JUST ALLOCATED? Y
MAXIMUM NUMBER OF DATA SETS: 6
DO YOU WANT WRITE VERIFY FOR THIS VOLUME? N
VOLUME INITIALIZED
ALLOCATE ANOTHER VOLUME? Y
VOLUME: VOL2
SIZE IN RECORDS: 200
VOL2     ALLOCATED
INITIALIZE THE VOLUME JUST ALLOCATED? Y
MAXIMUM NUMBER OF DATA SETS: 6
DO YOU WANT WRITE VERIFY FOR THIS VOLUME? N
VOLUME INITIALIZED
ALLOCATE ANOTHER VOLUME? Y
VOLUME: VOL3
SIZE IN RECORDS: 400
VOL3     ALLOCATED
INITIALIZE THE VOLUME JUST ALLOCATED? Y
MAXIMUM NUMBER OF DATA SETS: 12
DO YOU WANT WRITE VERIFY FOR THIS VOLUME? N
VOLUME INITIALIZED
ALLOCATE ANOTHER VOLUME? N

COMMAND (?):
```

Figure 24. Initialize a multivolume diskette

$INITDSK - Initialize Direct Access Device Volume Control *(continued)*

Use the II command to write the IPL text on a disk or diskette volume (the first sector of the disk or diskette). IPL text will then point to the IPL target (system nucleus) from which you wish to IPL. ayou may use any nucleus as the IPL target. This allows you to define more than one supervisor; however, the IPL text includes only one supervisor's location. When you write IPL text with the II command on a specific volume, the system replaces the previous IPL text. $INITDSK prompts you for the following information:

- The name of the system nucleus you want used as the IPL target. You can specify the full name of the nucleus, for example, $EDXNUCX (where x is any alphanumeric character) or only the last character of the nucleus name (for example, x for $EDXNUCX).
- The name of the volume containing the IPL target nucleus. The system writes IPL text on the disk or diskette containing this volume.

The following two examples show initializing IPL text. The first example shows specifying the full name of the system nucleus ($EDXNUCT); the second shows specifying only the last character of the nucleus name (T). The results are the same in each example.

Example 1: Initialize IPL text; specify full name of nucleus.

```
COMMAND (?): II
NUCLEUS: $EDXNUCT
VOLUME: EDX002
IPL TEXT WRITTEN

COMMAND (?):
```

Example 2: Initialize IPL text; specify only last character of nucleus name (T).

```
COMMAND (?): II
NUCLEUS: T
VOLUME: EDX002
IPL TEXT WRITTEN

COMMAND (?):
```

Notes:

1. When specifying the name of the volume you want initialized, specify EDX002 for disk or multivolume diskettes; for single-volume diskettes, specify your volume ID.
2. If you define the IPL nucleus to be other than $EDXNUC (allocated by $INITDSK), the system may move the nucleus if you compress ($COMPRES) the volume or the device. The IPL text then points to the wrong address unless you use II to rewrite the IPL text on the volume. Also, if the system moves $LOADER, you must use the starter system from diskette to load $INITDSK.

$INITDSK - Initialize Direct Access Device Volume Control *(continued)*

IV — Initialize Volume

Use the IV command to initialize (write) a data set directory in a disk or diskette volume. You have allocated the volume using the AV command. $INITDSK prompts for the following information:

- The name (1 to 6 characters) of the volume you want initialized.
- The number (in decimal) of data sets you want in the volume.
- Whether or not you wish to allocate $EDXNUC on the volume.

$INITDSK only prompts to allocate $EDXNUC if the volume is large enough to contain the system nucleus. If you respond Y, $INITDSK copies the current $EDXNUC from volume EDX002. If you have $EDXNUC allocated, $INITDSK also asks if you want the IPL text initialized. If you respond Y, it writes the IPL text on the volume and points to the system nucleus that it just copied. This allows you to IPL from this volume. If you respond N, the system does not initialize the IPL text. To be able to point at a later time to the system nucleus just copied, use the II command.

Once you use IV, you can no longer access the data that was on the volume before.

Note: After you issue an IV to a diskette on a 4964 device, you should open then close the door of the 4964.

Example: Initialize a volume containing $EDXNUC.

```
COMMAND (?): IV
VOLUME: EDX002
MAXIMUM NUMBER OF DATA SETS: 500
ALLOCATE $EDXNUC? Y
VOLUME INITIALIZED
INITIALIZE IPL TEXT? Y
IPL TEXT INITIALIZED

COMMAND (?):
```

$INITDSK - Initialize Direct Access Device Volume Control *(continued)*

LAV — List All Volumes

Use the LAV command to list the volume(s) on all disks or diskettes on the Series/1 and the number of records in each volume. LAV first lists information for the device you are using currently and then lists other devices.

$INITDSK prompts you for the following information:

- Whether or not you want to list the directory information.
- Whether or not you want to list the free space chain.

EDX uses the directory to keep track of the amount of free space available on a volume. You can list available free space, if it exists, by responding Y to the LIST FREE SPACE CHAIN? prompt.

You can direct the list to a printer or display station other than the one you are currently using by specifying the address of the target device.

$INITDSK - Initialize Direct Access Device Volume Control *(continued)*

Example: List all volumes.

Note: FIRST REC is the number of the record containing the first record of the volume, relative to the directory.

```
COMMAND (?): LAV

DEVICE ADDRESS: 0003

VOLUME   FIRST REC.       SIZE

EDX002        121        11000 **IPL=$EDXNUCT** **PERFORMANCE VOLUME**
EDX003      11121         9000
ASMLIB      20121        10000 **PERFORMANCE VOLUME**
EDX40       30121         7000
PRGRMS      37121         2500

LIST DIRECTORY INFO? Y

DEVICE SIZE(RECORDS):            54000
DIRECTORY REC. NO.:                361
DIRECTORY SIZE(RECS):              120
UNUSED DIRECTORY BYTES:          30424
NO. VOLUME ENTRIES:                  6
MAX. VOLUME ENTRIES:               958
FREE SPACE ENTRIES:                  1
DEVICE HAS NO FIXED HEADS

LIST FREE SPACE CHAIN? Y

  FIRST RECORD      SIZE
       45771        7870

DEVICE ADDRESS: 0002
SINGLE-VOLUME DISKETTE
EDX001       2220 RECORDS
COMMAND (?): EN
```

$INITDSK - Initialize Direct Access Device Volume Control *(continued)*

LP — List Performance Volumes

Use the LP command to list only volumes designated as performance volumes. You can indicate the device whose volumes you want listed by entering the command and the device address (in hexadecimal).

Example: List performance volumes for disk at device address 03.

```
COMMAND (?): LP 03

VOLNAME   DEVICE     VDE
          ADDRESS    ADDRESS

EDX002       0003       076A
ASMLIB       0003       0798

COMMAND (?):
```

LV — List Volumes

Use the LV command to list the volume(s) on a disk or diskette and the number of records in each volume. $INITDSK prompts you for the following information:

- The address (in hexadecimal) of the device.
- Whether or not you want to list the directory information.
- Whether or not you want to list the free space chain.

EDX uses the directory to keep track of the amount of free space. If free space exists on the volume, you can find it listed under the directory information. $INITDSK issues the LIST FREE SPACE CHAIN? prompt so you can list the location of the free space and its size (number of 256-byte records).

You can direct the list to a printer or display station other than the one you are currently using by specifying the address of the target device.

$INITDSK - Initialize Direct Access Device Volume Control *(continued)*

Example 1: List volumes on the disk at address 03.

Note: FIRST REC is the number of the record containing the first record of the volume, relative to the directory.

```
COMMAND (?): LV
DEVICE ADDRESS: 03

DEVICE ADDRESS: 0003

VOLUME   FIRST REC.       SIZE

SUPLIB          121       2500 **PERF. VOL**
ASMLIB         2621      10000 **PERF. VOL**
EDX002        12621      23380 **IPL=$EDXNUC ** **PERF. VOL**
FHVOL             1        480 **FHVOL** **PERF. VOL**

LIST DIRECTORY INFO? Y

DEVICE SIZE(RECORDS):          36240
DIRECTORY REC. NO.:              241
DIRECTORY SIZE(RECS):            120
UNUSED DIRECTORY BYTES:        30564
NO. VOLUME ENTRIES:                4
MAX. VOLUME ENTRIES:             958
FREE SPACE ENTRIES:                0
DEVICE HAS FIXED HEADS

FIXED HEAD-VOLUME IS ALLOCATED

COMMAND (?): EN
```

$INITDSK - Initialize Direct Access Device Volume Control *(continued)*

Example 2: List volumes on disk at address 48.

This list volume example corresponds to Figure 22 on page UT-408 in which the user moved the directory to a record other than the default. In addition, this example lists the free space available on the volume. FIRST REC is the number of the first record within the free space and SIZE is the amount of free space (number of 256-byte records) starting at the FIRST REC.

```
COMMAND (?): LV
DEVICE ADDRESS: 48

DEVICE ADDRESS:  0048

VOLUME   FIRST REC.       SIZE

FHVL49            1        512 **FH  VOL**
VOL1            121      10000 WRITE VERIFY
VOL2          10121      20000 **IPL=$EDXNUC **

LIST DIRECTORY INFO? Y

DEVICE SIZE(RECORDS):          229120
DIRECTORY REC. NO.:             50049
DIRECTORY SIZE(RECS):             120
UNUSED DIRECTORY BYTES:         30584
NO. VOLUME ENTRIES:                 3
MAX. VOLUME ENTRIES:              958
FREE SPACE ENTRIES:                 1
DEVICE HAS FIXED HEADS

FIXED-HEAD VOLUME IS ALLOCATED

LIST FREE SPACE CHAIN? Y

  FIRST REC.        SIZE
       30121      148952

COMMAND (?): EN
```

$INITDSK - Initialize Direct Access Device Volume Control *(continued)*

RD — Rename Diskette VOL1/OWNERID

Use the RD command to rename a volume label (VOL1) and owner id on a diskette. You can use individual diskettes on different diskette devices on the same or a different Series/1. Therefore, you should assign each a unique name called a volume label. Owner identification indicates who owns the contents of the diskette. When you rename the volume label and owner id on a diskette, you can no longer access the data under the old name.

$INITDSK supplies the current volume label and the current owner ID and prompts you for your new volume label and owner ID.

Example: Rename an existing volume label and owner identification.

```
COMMAND (?): RD 2

CURRENT VOLUME LABEL: MYVOL
ENTER VOLUME LABEL: NEWVOL

CURRENT OWNER ID: MYDSKT
ENTER OWNER ID: JWW

COMMAND (?):
```

If you enter the address of a device other than a diskette, $INITDSK issues the following message:

```
ONLY VALID FOR A DISKETTE

COMMAND (?):
```

RV — Rename Volumes

Use the RV command to rename a volume on a disk or multivolume diskette device. When you rename a volume, you can no longer access the data under the old name. You can rename a volume with a name that exists on another disk or diskette on the system, but this is not a recommended procedure. This should be used only as the last step before you remove your diskette. $INITDSK prompts you for the address of the volume you want renamed, the current volume name, and the new volume name.

Example: Rename a volume.

```
COMMAND (?): RV
DEVICE ADDRESS: 48
VOLUME: SRCLIB
NEWNAME: SOURCE
RENAME COMPLETED
```

$INITDSK - Initialize Direct Access Device Volume Control *(continued)*

SV — Split Volume

Use the SV command to split an existing disk volume or multivolume diskette volume and to allocate an additional volume or volumes in the free space at the end of the existing volume. With the option of defining additional volumes, the SV command enables you to utilize the entire disk. Splitting a volume does not alter existing data.

To define additional volumes on a disk device, the free space must be contiguous and at the end of the volume. If the available free space is fragmented, use $COMPRES to compress the volume before using the SV command. If the size of the volume (in records) specified on the SV command exceeds the space at the end of the volume, the system issues a message and does not allocate the new volume.

To determine if you have free space on a disk device and if that free space is contiguous or fragmented, use either of the following commands:

- LV command of $INITDSK
- LS command of $DISKUT1

You cannot split a volume on a single-volume diskette device.

Example 1: Split a volume with sufficient free space at the end of the volume.

```
COMMAND (?): SV
VOLUME: EDX002
NEW VOLUME: EDX003
SIZE IN RECORDS: 10000
EDX002 SPLIT    EDX003 ALLOCATED

COMMAND (?):
```

Example 2: Split a volume without sufficient free space at the end of the volume.

```
COMMAND (?): SV
VOLUME: EDX002
NEW VOLUME: EDX003
SIZE IN RECORDS: 10000
INSUFFICIENT CONTIGUOUS SPACE AT END OF VOLUME

COMMAND (?):
```

$INITDSK - Initialize Direct Access Device Volume Control *(continued)*

VD — Verify Device

Use the VD command to verify that each record in a disk or diskette device is readable. The system locates defective records so that you can assign alternates with the $DASDI utility. The system issues a message for each defective record it locates. Verifying records on a disk or diskette device does not modify the data currently on the device. $INITDSK prompts you for the address (in hexadecimal) of the device you want verified.

Example: Verify a device.

```
COMMAND (?): VD
DEVICE ADDRESS: 48
    229632 RECORDS VERIFIED

COMMAND (?):
```

VV — Verify Volume

Use the VV command to verify that each record in a disk or diskette volume is readable. The system locates defective records so that you can assign alternates with the $DASDI utility. The system issues a message for each defective record it located. Verifying records on a disk or diskette volume does not modify the data currently on the volume.

$INITDSK prompts you for the following information:

- The name of the volume you want verified.
- Whether or not you wish to verify the entire volume.
- The number (in decimal) of the first record you want verified.
- The total number (in decimal) of records you want verified.

$INITDSK - Initialize Direct Access Device Volume Control *(continued)*

Example 1: Verify the first 500 records on a volume.

```
COMMAND (?): VV
VOLUME: EDX002
VERIFY ENTIRE VOLUME? Y
         500 RECORDS VERIFIED

COMMAND (?):
```

Example 2: Verify a volume with defective records. The system issues a message for each defective record it locates.

```
COMMAND (?): VV
VOLUME: EDX002
VERIFY ENTIRE VOLUME? N
FIRST RECORD: 1
NUMBER OF RECORDS: 500
ERROR AT RECORD 10, DISK I/O RETURN CODE= 2
ERROR AT RECORD 16, DISK I/O RETURN CODE= 5
         500 RECORDS VERIFIED

COMMAND (?):
```

WN — Clear Write Verify

Use the WN command to remove the write verify set for a specific volume. When you clear write verify, EDX does not verify that the data written is correct.

Example: Clear write verify.

```
COMMAND: WN
VOLUME: MYVOL

COMMAND:
```

WV — Set Write Verify

Use the WV command to set the write verify on for a specific volume. By setting write verify, EDX rereads all data that it writes on the data sets on the volume to make sure the data was written correctly. You can set write verify off or clear it by using the WN command.

Example: Set write verify on.

```
COMMAND: WV
VOLUME: MYVOL

COMMAND:
```

$IOTEST - Test Sensor I/O; List Configuration

$IOTEST determines the complete I/O configuration of a Series/1 and tests the operation of sensor-based I/O features. $IOTEST performs the following functions:

- Digital reads and writes (group or subgroup)
- Digital writes with selected time intervals
- External sync DI and DO
- Interrupt processing (normal, special bit, and group)
- Analog reads and writes
- Hardware configuration listing of the Series/1
- Device listing (devices supported by the system).

Invoking $IOTEST

You invoke $IOTEST with the $L command or option 9.3 of the session manager.

$IOTEST - Test Sensor I/O List Configuration *(continued)*

$IOTEST Commands

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

```
> $L $IOTEST
LOADING $IOTEST     36P,13:26:42, LP=5000, PART=1

ATTLIST (ALTER) TO STOP LOOPING FUNCTIONS

COMMAND (?): ?

EN = END PROGRAM
DO = DIGITAL OUTPUT
PD = UP AND DOWN DO WITH TIME
XO = EXTERNAL SYNC DO
DI = DIGITAL INPUT
XI = EXTERNAL SYNC DI
PI = PROCESS INTERRUPT
SG = SPECIAL PROCESS INTERRUPT GROUP
SB = SPECIAL PROCESS INTERRUPT BIT
AI = ANALOG INPUT
AO = ANALOG OUTPUT
LD = LIST ALL HARDWARE DEVICES
LS = LIST CURRENT SUPERVISOR HARDWARE DEVICES
WS = PUT PROGRAM IN WAIT STATE
LISTP = DIRECT LISTING TO $SYSPRTR
LISTT = DIRECT LISTING TO TERMINAL

COMMAND (?):
```

After $IOTEST displays the commands, you are again prompted with COMMAND (?):. Then you can respond with the command of your choice (for example, DO). Press the attention key and enter ALTER to end repetitive functions and to reactivate the program if it enters the wait state. For example:

```
ENTER MPXR POINT (0-15):  1
AI VOLTAGE =    -629 MV, E-3
AI VOLTAGE =    -688 MV, E-3
> ALTER
COMMAND (?):
```

Each command and its explanation is presented in alphabetical order on the following pages.

$IOTEST - Test Sensor I/O List Configuration *(continued)*

AI — Analog Input

Use AI to read analog input. AI issues a read every 10 milliseconds and only prints the value if it is different than that of the last reading.

Example 1: Read analog input.

```
COMMAND (?):  AI
ENTER DEVICE ADDRESS, (HEX 1-FF):  61
ENTER RANGE: 1=5V, 2=500MV, 3=200MV, 4=100MV,
             5=50MV, 6=20MV, 7=10MV:  7
ZERO CORRECTION?  N
ENTER MPXR POINT (0-15):  1
AI VOLTAGE =    -629 MV, E-3
AI VOLTAGE =    -688 MV, E-3
> ALTER
COMMAND (?):
```

Analog input has a testing facility to convert diagnostic zero or voltage. $IOTEST allows these functions if you give the ADC address instead of the multiplexer address.

Example 2: Convert diagnostic zero or voltage.

```
COMMAND (?):  AI
ENTER DEVICE ADDRESS, (HEX 1-FF):  60
CONVERT DIAGNOSTIC ZERO?  Y
ENTER RANGE: 1=5V, 2=500MV, 3=200MV, 4=100MV,
             5=50MV, 6=20MV, 7=10MV:  1
AI VOLTAGE =       0 MV, E-0
> ALTER

COMMAND (?):  AI
ENTER DEVICE ADDRESS, (HEX 1-FF):  60
CONVERT DIAGNOSTIC ZERO?  N

CONVERTING DIAGNOSTIC VOLTAGE, SHOULD BE 4.5 +- 0.5
AI VOLTAGE =    4604 MV, E-0
AI VOLTAGE =    4602 MV, E-0
> ALTER
COMMAND (?):



COMMAND (?):  AI
ENTER DEVICE ADDRESS, (HEX 1-FF):  60
CONVERT DIAGNOSTIC ZERO?  N

CONVERTING DIAGNOSTIC VOLTAGE, SHOULD BE 4.5 +- 0.5
AI VOLTAGE =    4604 MV, E-0
AI VOLTAGE =    4602 MV, E-0
> ALTER
COMMAND (?):
```

$IOTEST - Test Sensor I/O List Configuration *(continued)*

DI — Digital Input

Use DI to read digital input.

Example: Read digital input.

```
COMMAND (?):  DI
ENTER DEVICE ADDRESS, (HEX 1-FF):  50
ENTER START BIT (0-15):  0
ENTER # OF BITS:  16
VALUE = A5A5
VALUE = C0FE
> ALTER
COMMAND (?):
```

DO — Digital Output

Use DO to write digital output.

Example: Write X'A5A5' to DO address 52

```
COMMAND (?):  DO
ENTER DEVICE ADDRESS, (HEX 1-FF):  52
ENTER START BIT (0-15):  0
ENTER # OF BITS:  16
ENTER DATA (HEX):  A5A5
COMMAND (?):
```

An easy way to test the system is to use the Customer Engineer's wrap-back connectors. The wrap cable for the IDIO unit connects the first DI address on the card to the first DO address and the same for the second DI and DO. These connections include the external sync functions, also. Therefore, you can execute two copies of $IOTEST simultaneously. Similar connectors are available for the 4982 Sensor I/O Unit.

EN — End $IOTEST

Use the EN command to end the $IOTEST utility.

Example: End $IOTEST.

```
COMMAND (?):  EN

$IOTEST ENDED AT 13:27:29
```

$IOTEST - Test Sensor I/O List Configuration *(continued)*

LD — List Devices

Use LD to list the devices attached to your Series/1. LD reads the actual hardware addresses and their IDs and displays a list of the descriptions. If a device exists but is not powered on, the system still displays the description for that device. You can direct the listing to the terminal that invoked $IOTEST or to the $SYSPRTR by using either the LISTT or LISTP command.

Example: List Series/1 hardware configuration.

```
COMMAND (?):  LD

ACTUAL SERIES/1 HARDWARE CONFIGURATION

ADDRESS   ID     DEVICE TYPE

  01     0206    4974 PRINTER
  02     0106    4964 DISKETTE UNIT
  03     00BA    4962 DISK MODEL 1F OR 2F WITH FIXED HEADS
  04     0406    4979 DISPLAY STATION
  09     1006    SINGLE-LINE BISYNC
  24     040E    4978 DISPLAY STATION
  40     0028    TIMER FEATURE
  41     0028    TIMER FEATURE
  44     5152    DDSK DISK/DISKETTE UNIT
  45     5212    4965 DISKETTE UNIT
  48     220E    FOUR LINE MLTA (2091/2092)
  49     220E    FOUR LINE MLTA (2091/2092)
  4A     220E    FOUR LINE MLTA (2091/2092)
  4B     220E    FOUR LINE MLTA (2091/2092)
  58     3106    4963 DISK SUBSYSTEM
  80     0402    SDLC MULTIDROP I/O ATTACHMENT
  81     0402    SDLC MULTIDROP I/O ATTACHMENT
  82     0402    SDLC MULTIDROP I/O ATTACHMENT
  83     0402    SDLC MULTIDROP I/O ATTACHMENT
  84     0402    SDLC MULTIDROP I/O ATTACHMENT
  85     0402    SDLC MULTIDROP I/O ATTACHMENT
  86     0402    SDLC MULTIDROP I/O ATTACHMENT
  87     0402    SDLC MULTIDROP I/O ATTACHMENT
  C0     232E    ALPA MULTIPORT I/O ATTACHMENT
  C1     232E    ALPA MULTIPORT I/O ATTACHMENT
  C2     232E    ALPA MULTIPORT I/O ATTACHMENT
  C3     232E    ALPA MULTIPORT I/O ATTACHMENT
  C4     232E    ALPA MULTIPORT I/O ATTACHMENT
  C5     232E    ALPA MULTIPORT I/O ATTACHMENT
  C6     232E    ALPA MULTIPORT I/O ATTACHMENT
  C7     232E    ALPA MULTIPORT I/O ATTACHMENT

COMMAND (?):
```

$IOTEST - Test Sensor I/O List Configuration *(continued)*

LS — List Current Supervisor Hardware Devices

LS provides a display similar to LD except that it lists only the devices your supervisor supports after system IPL; this list can be different from the actual hardware attached to your Series/1. You can direct the listing to the terminal that invoked $IOTEST or to the $SYSPRTR by using either the LISTT or LISTP command.

Note: The system will not list communication device cards that you did not install at IPL time.

Example: List devices supported by supervisor.

```
COMMAND (?):  LS

HARDWARE DEVICES SUPPORTED BY THIS SUPERVISOR

ADDRESS   ID     DEVICE TYPE

 01      0206    4974 PRINTER
 02      0106    4964 DISKETTE UNIT
 03      00BA    4962 DISK MODEL 1F OR 2F WITH FIXED HEADS
 24      040E    4978 DISPLAY STATION
 44      5152    DDSK DISK/DISKETTE UNIT
 80      0402    4980 DISPLAY STATION
 C0      2003    5225 PRINTER
 C1      2001    5224 PRINTER

COMMAND (?):
```

PD — Up and Down Digital Output with Time.

Use PD to write digital output and specify various time intervals.

Example: Using DO address 53, set bit 8 on for 10 milliseconds, off for 50 milliseconds, and do this 100 times.

```
COMMAND (?):  PD
ENTER DEVICE ADDRESS ,(HEX 1-FF):  53
ENTER START BIT (0-15):  8
ENTER # OF BITS:  1
ENTER DATA1 (0-FFFF):  1
ENTER TIME1 IN MS:  10
ENTER DATA2 (0-FFFF):  0
ENTER TIME2 IN MS:  50
ENTER NUMBER OF TIMES TO LOOP:  100
COMMAND (?):
```

Note: If you set ENTER NUMBER OF TIMES TO LOOP less than or equal to zero, the looping continues until you enter the ALTER command.

$IOTEST - Test Sensor I/O List Configuration *(continued)*

PI — Process Interrupt

Use PI to test for the occurrence of the process interrupt.

Example: Test process interrupt for the occurrence of event.

```
COMMAND (?):  PI
ENTER DEVICE ADDRESS, (HEX 1-FF):  50
ENTER BIT (0-15):  3
PI OCCURRED
PI OCCURRED
PI OCCURRED
> ALTER
COMMAND (?):
```

SB/SG — Special Process Interrupt Bit/Group

Functionally, the SB and SG commands operate differently within the supervisor, but they print basically the same information as normal PI with this utility program.

XI — External Sync for Digital Input

Use XO to read digital input using external synchronization.

Example: Read digital input using external sync.

```
COMMAND (?):  XI
ENTER DEVICE ADDRESS, (HEX 1-FF):  51
ENTER WORD COUNT:  100
COMMAND (?):
```

XO — External Sync for Digital Output

Use XO to write digital output using external synchronization.

Example: Write digital output using external sync.

```
COMMAND (?):  XO
ENTER DEVICE ADDRESS, (HEX 1-FF):  53
ENTER WORD COUNT 1-256:  100
COMMAND (?):
```

Note: The system writes data from a buffer within this program that is used for external sync DI and DO. Therefore, you can input data by using DI and output data by using DO.

$JOBQUT - Controlling Job Queue Processing

$JOBQUT controls the job queue processing as a whole. Its commands perform actions on the entire job queue processing system rather than on individual jobs. (To submit and control individual jobs, see "$SUBMIT - Submit/Control Jobs in Job Queue Processor" on page UT-503.)

The job queue processing utility enables you to do the following:

- Suspend and resume job queue processing.
- Change the logging terminal.
- Delete all jobs in the queue.
- Display the status of the job queue processing system.
- Reinitialize job queue data set.
- Terminate job queue processing.

Note: Do not use $JOBQ to run interactive programs.

Invoking $JOBQUT

You invoke $JOBQUT with the $L command or option 10.1 of the session manager.

```
 >  $L $JOBQUT

LOADING $JOBQUT     8P,09:18:62, LP=2300, PART=1

$JOBQUT - EDX JOB QUEUE CONTROLLER PROGRAM

ENTER '?' AT ANY TIME FOR HELP

COMMAND (?):
```

After you invoke $JOBQUT, you can enter any of the available commands. If this is the first time you enter a command and the job queue processor has not been used previously, $JOBQUT issues the following messages:

```
JOB QUEUE PROCESSOR ($JOBQ) MUST BE LOADED TO CONTINUE.  LOAD IT (Y/N)?

PARTITION NUMBER (1-8) OR ENTER FOR ANY:
```

Note: We recommend that you always check the status of the job queue processor and job queue processing. To check the status of the job queue processor, enter the STQ command in response to the COMMAND (?): prompt. (See "STQ — Display the Status of Job Queue Processing" on page UT-435 for a description of theSTQ command.)

$JOBQUT - Controlling Job Queue Processing *(continued)*

The job queue processor, $JOBQ, starts if you respond Y to the loading prompt. $JOBQUT prompts you for a partition number in which to load $JOBQ. However, you do not need to select a partition; just press the enter key, and the supervisor selects a partition large enough to contain $JOBQ. $JOBQUT issues the following message:

```
JOB QUEUE PROCESSOR LOADED INTO PARTITION x
```

where x indicates the partition where the job queue processor is loaded.

$JOBQUT then issues the following messages indicating its own status, the logging terminal, and if there are jobs in the job queue.

```
JOB QUEUE PROCESSING HAS BEEN SUSPENDED

LOGGING TERMINAL IS $SYSPRTR

THERE ARE NO JOBS IN THE QUEUE

COMMAND (?):
```

$JOBQUT Commands

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

```
COMMAND (?):  ?

CLT - CHANGE LOGGING TERMINAL
DJQ - DELETE JOB QUEUE
RJQ - RESUME JOB QUEUE PROCESSING
SJQ - SUSPEND JOB QUEUE PROCESSING
STQ - DISPLAY STATUS OF JOB QUEUE PROCESSING
TJQ - TERMINATE JOB QUEUE PROCESSING
DCH - DETACH $JOBQUT; RESUME-ATTN JOBQUT,END-ATTN ENDJUT
IJQ - REINITIALIZE JOB QUEUE DATA SET (JOBQDS)

EN  - END $JOBQUT

COMMAND (?):
```

After $JOBQUT displays the commands, it prompts you with COMMAND (?): again. Then you can respond with the command of your choice (for example, RJQ). Each command and its explanation is presented in alphabetical order on the following pages.

$JOBQUT - Controlling Job Queue Processing *(continued)*

CLT — Change Logging Terminal

Use the CLT command to change the terminal where the status (the start and end) of jobs is displayed. Initially, the job queue logging terminal is $SYSPRTR. Once you specify the new logging terminal, you have the choice of placing the terminal in roll screen mode. If you answer Y and turn on roll screen mode, you do not need to press the enter key after the screen is full. Output "rolls" off the top of the screen, as new terminal output appears at the bottom of the screen. If you answer N, the roll option remains off. $JOBQ stops processing and you need to press the enter key each time the screen fills up to display further terminal output.

Notes:

1. If you are running a large number of jobs, we recommend turning roll screen mode on.
2. The roll screen mode option does not apply to printers; however, $JOBQUT still issues the roll mode prompt. Enter Y or N to allow $JOBQUT to continue.

Example: Change the logging terminal and turn on roll screen mode.

```
COMMAND (?):   CLT

THE LOGGING TERMINAL IS NOW $SYSPRTR
NEW LOGGING TERMINAL NAME OF '*' FOR THIS TERMINAL:   $SYSLOG

LOGGING TERMINAL CHANGED TO $SYSLOG

PUT TERMINAL IN ROLL MODE (Y/N)?  Y

COMMAND (?):
```

Log messages are directed to the terminal specified until you change the logging terminal. If you terminate and restart job queue processing, log messages are still directed to the terminal last specified. If you do not wish to change the logging terminal, press the enter key.

$JOBQUT - Controlling Job Queue Processing *(continued)*

DCH — Detach $JOBQUT

Use the DCH command to place $JOBQUT in suspended mode. To resume operation, press the attention key and type JOBQUT

To end $JOBQUT without resuming operation, press the attention key and enter ENDJUT.

Example:

```
COMMAND (?):  DCH

>JOBQUT

COMMAND (?):  DCH

>ENDJUT

$JOBQUT ENDED AT 00:12:03
```

DJQ — Delete Job Queue

Use the DJQ command to delete all jobs currently in the job queue. Once you enter DLQ, $JOBQUT questions whether you want to delete all jobs in the queue. Enter Y to delete all jobs, N to cancel the command.

Example: Delete jobs in job queue.

```
COMMAND (?):  DJQ

DELETE ALL JOBS IN THE QUEUE (Y/N)?  Y

JOB QUEUE DATA SET REINITIALIZED

COMMAND(?):
```

EN — End $JOBQUT

Use the EN command to end the $JOBQUT utility.

Example: End the $JOBQUT utility.

```
COMMAND (?):  EN

$JOBQUT ENDED
```

$JOBQUT - Controlling Job Queue Processing *(continued)*

IJQ — Reinitialize Job Queue Data Set (JOBQDS)

Use the IJQ command to initialize the job queue data set (JOBQDS) if the system detects an error. If the job queue data set has an error, $JOBQUT issues the following message:

```
READ/WRITE ERROR IN JOBQDS
```

The logging terminal defaults to $SYSPRTR Use the CLT command to change the logging terminal.

Example: Initialize the job queue data set.

```
COMMAND (?):   IJQ
JOB QUEUE DATA SET REINITIALIZED
```

RJQ — Resume Job Queue Processing

Use the RJQ command to resume job queue processing after you have suspended processing with the SJQ command. When the job queue processor is initially loaded, it is in suspended mode; use the RJQ command to activate it.

Example: Resume job queue processing.

```
COMMAND (?):   RJQ
JOB QUEUE PROCESSING HAS BEEN RESUMED
COMMAND (?):
```

SJQ — Suspend Job Queue Processing

Use the SJQ command to suspend job queue processing. All jobs in the job queue are held. However, any job that is already executing when you issue the SJQ command will run to completion. Jobs submitted for processing after job queue processing is suspended are held in the job queue until job queue processing is resumed.

Example: Suspend job queue processing.

```
COMMAND (?):   SJQ
JOB QUEUE PROCESSING HAS BEEN SUSPENDED
COMMAND (?):
```

$JOBQUT - Controlling Job Queue Processing *(continued)*

STQ — Display the Status of Job Queue Processing

Use the STQ command to display the status of job queue processing on your display terminal. If job queue processing is in effect and there are jobs in the job queue, $JOBQUT issues the following messages:

```
JOB QUEUE PROCESSING IS IN EFFECT

LOGGING TERMINAL IS xxxxxxx

JOB ID | COMMAND DATA SET | PRIORITY | STATUS

  2      OVERNITE,USRVOL       2        HELD
  3      CALCJOB ,PAYROL       1        WAITING
  4      CHECKS  ,PAYROL       0        EXECUTING
  5      CALCJOB ,EDX002       2        WAITING
  6      JOB1    ,EDX002       2        WAITING
  7      ASMJOB1 ,EDX002       3        HELD

NUMBER OF JOBS QUEUED IS    6

COMMAND (?):
```

If $JOBQUT detects an error in the job queue data set, it issues the following messages:

```
INTERNAL ERROR DETECTED IN JOB QUEUE DATA SET

THE JOB QUEUE DATA SET MUST BE REINITIALIZED BY
THE "IJQ" COMMAND IN $JOBQUT
```

$JOBQUT - Controlling Job Queue Processing *(continued)*

TJQ — Terminate Job Queue Processing

Use the TJQ command to end job queue processing. You are asked if the jobs currently in the job queue should be deleted. If you respond Y, all jobs are deleted. If you respond N, $JOBQ maintains the job queue and the jobs in the queue execute when job queue processing restarts. When you terminate job queue processing, any job that is executing will complete. In addition, $JOBQ (the job queue processor) and $JOBQUT end.

Example: End job queue processing.

```
COMMAND (?):  TJQ

TERMINATE JOB QUEUE PROCESSING (Y/N)?  Y

DELETE ALL JOBS IN THE QUEUE (Y/N)?  N

$JOBQ ENDED AT 10:11:62

JOB QUEUE PROCESSING TERMINATED

$JOBQUT ENDED
```

Note: You can only terminate job queue processing — and $JOBQ — with this command. You cannot cancel $JOBQ with the $C operator command. If any job in the job queue is executing, $JOBQ will not end.

$JOBUTIL - Job Stream Processor

$JOBUTIL is a batch processing capability that you can invoke simultaneously with the execution of other programs. This allows you to execute a series of programs sequentially without intervention. You include the names of the programs and other information in a collection of $JOBUTIL commands that you create with $EDIT1N or $FSEDIT.

A procedure can invoke another procedure; however, the called procedure cannot invoke another procedure. A typical use of $JOBUTIL would be to execute a procedure that causes the assembly, link-editing, and formatting of your program. Refer to "Batch Job Example" on page UT-455. For additional examples of using $JOBUTIL, see the *Operation Guide*.

$JOBUTIL is the main program. It loads and waits for the termination of the following three independent programs that actually carry out the majority of the work:

- $JP1 - opens required data sets
- $JP2 - interprets all commands
- $DISKUT3 - executes the AL and DE commands

Programs that are capable of receiving parameters in the format used by $JOBUTIL are:

```
$COBOL          $PASCAL
$EDXASM         $PL/I
$FORT           $PREFIND
$LINK           $S1ASM
$EDXLINK        $UPDATE
$EDXLIST
```

Invoking $JOBUTIL

You invoke $JOBUTIL with the $L operator command or option 7 of the session manager.

Setup Procedure

To create your procedure, use either of the text editing facilities, $EDIT1N or $FSEDIT. If you use $EDIT1N to create your procedure, you must use $DISKUT1 to preallocate a disk or diskette data set to save the procedure. If you use $FSEDIT to create your procedure, you do not have to preallocate a data set. $FSEDIT automatically creates it for you.

To run your job, load $JOBUTIL and specify the name of your procedure data set.

```
> $L $JOBUTIL
DS1 (NAME,VOLUME):
```

Refer to the *Operation Guide* for instructions on setting up your own JOBUTIL procedure.

$JOBUTIL - Job Stream Processor *(continued)*

$JOBUTIL Commands

The $JOBUTIL commands are listed below. Commands must be entered in the following format:

Command - Start in position 1.

Operands - Start in position 10.

Comment - Start in position 18 or after. You can extend a comment as far as position 71. (At least one blank must separate the comment from the operands.) If a command does not have any operands, its comments can start in column 10.

For internal comments, use '*' in position 1

Command	Function
AL	Allocate data set
DE	Delete data set
DS	Identify data set
EJECT	Start a new page in log listing
EOJ	Identify end of job
EOP	Identify end of nested procedure
EXEC	Load and execute program
JOB	Identify job
JUMP	Jump to label
LABEL	End jump
LOG	Log Job Stream Processor Commands
NOMSG	Suppress load messages
PARM	Identify program parameter
PAUSE	Await manual intervention
PROC	Identify nested procedure
PROGRAM	Identify program
REMARK	Remark to operator
*	Internal comment

$JOBUTIL - Job Stream Processor *(continued)*

AL — Allocate Data Set

Use the AL command to identify a data set to be allocated. It returns a $DISKUT3 return code that can be used by the JUMP command.

Note: An AL statement between a PROGRAM and EXEC statement can cause unpredictable results.

Syntax:

```
AL        name,volume size type

Required: name,volume
          size
          type
Default:  none
```

Operands	***Description***
name	Defines the data set to be allocated.
volume	Defines the volume on which the data set is to be allocated.
size	Defines the size in blocks of the data set.
type	Defines the type of data set as follows: D indicates data (default) P indicates program

Example:

```
AL        TEMPDS,EDX003 25 D
```

$JOBUTIL - Job Stream Processor *(continued)*

DE — Delete Data Set

Use the DE command to identify a data set to be deleted.

Note: A DE statement between a PROGRAM and EXEC statement can cause unpredictable results.

Syntax:

```
 DE       name,volume

Required: name,volume
Default:  none
```

Operands	*Description*
name	Defines the data set to be deleted.
volume	Defines the volume from which the data set is to be deleted.

Example:

```
DE        TEMPDS,EDX003
```

$JOBUTIL - Job Stream Processor *(continued)*

DS — Identify Data Set

Use the DS command to identify a data set to be opened and accessed by a program. DS commands are allowed only between PROGRAM and EXEC commands. Up to nine (9) DS commands can be specified for a program.

Syntax:

```
DS       name,volume

Required: name
Default:  volume is IPL volume
```

Operands	***Description***
name	Defines the data set to be accessed by a program.
volume	Defines the volume that contains the data set to be accessed by a program.

Example:

```
DS          WORK1,EDX001
```

$JOBUTIL - Job Stream Processor *(continued)*

EJECT — Start New Page in Log Listing

Use the EJECT command to print the next command in the log listing at the top of a new page. If no logging is being done, EJECT commands are ignored.

Syntax:

```
 EJECT

Required:  none
Default:  none
```

Operands *Description*

None

Example:

```
EJECT
```

$JOBUTIL - Job Stream Processor *(continued)*

EOJ — End of Job

Use the EOJ command to indicate the end of the primary procedure. EOJ must be the last command in the procedure data set.

Syntax:

```
EOJ

Required:  none
```

Operands	***Description***
None	

EOP — End of Procedure

Use the EOP command to indicate the end of a nested procedure. EOP must be the last command in the called procedure data set.

Syntax:

```
EOP

Required:  none
```

Operands	***Description***
None	

$JOBUTIL - Job Stream Processor *(continued)*

EXEC — Execute Program

Use the EXEC command to load and execute the program identified in the preceding PROGRAM command. EXEC must have been preceded by a PROGRAM command.

Syntax:

```
 EXEC     STORAGE=size
Required: none
Default:  STORAGE=0
```

Operands	*Description*
STORAGE	Specifies the amount of dynamic storage to be allocated to the program to be executed. This value overrides the value specified when the program was compiled. STORAGE=0 (the default) indicates that the dynamic storage specified (if any) during compilation should be allocated.

JOB — Identify Jot

Use the JOB command, an optional command, to identify the procedure or a collection of commands within a procedure data set. When a JOB command is read by $JOBUTIL, a message is printed on the terminal with the job name, time started, and date (if timer support is included).

Syntax:

```
 JOB      job-name

Required: job-name
```

Operands	*Description*
job-name	Names the current group of commands as a job.

Example:

```
JOB          TEST1
```

$JOBUTIL - Job Stream Processor *(continued)*

JUMP — Jump to Label

Use the JUMP to bypass $JOBUTIL commands by testing the completion code of the previously executed program. The test compares the program completion code to ‘cc’. The commands between the JUMP and the LABEL command with the same label name are ignored if the condition tested for is ‘true’. An unconditional JUMP to a LABEL is also permitted. Jumps are forward only and are limited to the range of the current procedure; for example, a jump cannot occur from one procedure to a label in another procedure. A JUMP with no label name jumps to the next unlabeled LABEL command or to an EOJ or an EOP, whichever occurs first. If the system encounters an EOJ or an EOP while searching for a LABEL command, it stops its search for the LABEL command and executes the EOJ or EOP.

Note: The JUMP and LABEL commands are not permitted in the PAUSE mode.

Syntax:

```
JUMP     label-name,op,cc

Required: none
```

Operands	*Description*
label-name	The name associated with a LABEL command.
op	LT, GT, EQ, GE, LE, or NE.
cc	Program completion code.

Examples:

```
JUMP       LBL1,EQ,20
JUMP       LBL2
JUMP
```

$JOBUTIL - Job Stream Processor *(continued)*

LABEL — Identify Continuation Point

Use the LABEL command to continue job processing after a related JUMP command is encountered. Job processing continues at the LABEL command. The LABEL command is not valid in the PAUSE mode

Syntax:

```
LABEL    label-name

Required:  none
```

Operands	*Description*
label-name	Defines the command where processing can continue.

Examples:

```
LABEL       LBL1
LABEL
```

$JOBUTIL - Job Stream Processor *(continued)*

LOG — Log Control

Use the LOG command to indicate whether the $JOBUTIL commands are to be printed as they are read and which terminal device is to print them. You can place LOG commands anywhere in the procedure data set. If you do not specify a terminal device, the commands are logged on the terminal from which you invoked the Job Processor.

Syntax:

```
LOG      Operand

Required: none
Default:  ON
```

Operands	*Description*
ON	Indicates that $JOBUTIL commands are to be printed.
OFF	Indicates that $JOBUTIL commands are not to be printed.
terminal-name	Identifies the terminal device which is to print the job processor commands. If you omit this operand, if omitted indicates that the job processor commands are to be logged on the terminal from which the job processor was called.

Examples:

```
LOG        ON
LOG        OFF
LOG        $SYSPRTR
LOG
```

$JOBUTIL - Job Stream Processor *(continued)*

NOMSG — No Message Logging

Use the NOMSG command to set LOGMSG=NO for the program identified in the preceding PROGRAM command. See the LOAD instruction in the *Language Reference* for the definition of LOGMSG. NOMSG is invalid if not preceded by a PROGRAM command.

The NOMSG command must be between the PROGRAM and EXEC commands.

Syntax:

```
NOMSG

Required: none
```

Operands	*Description*
None	

Example:

```
PROGRAM    CALCDEMO
NOMSG
```

$JOBUTIL - Job Stream Processor *(continued)*

PARM — Pass Parameter

Use the PARM command to identify the parameters being passed to the program in the preceding PROGRAM command. PARM must be between the PROGRAM and EXEC commands. Maximum length of the operand on the PARM command is 62 characters. The parameters specified on the PARM command are passed to the specified program as an EBCDIC character string. In the specified program they can be referenced as beginning at the label $PARM1 and are packed two characters per word. The length of the string is determined by the PARMN operand of the PROGRAM statement in the specified program. For example, PARM=31 would cause the maximum 62 characters from positions 10-71 of the PARM command to be transferred to the specified program starting at the label $PARM1.

Syntax:

```
PARM     parameters

Required: none
```

Operands	***Description***
parameters	Defines parameters to be passed to the program.

Example:

```
PARM      NOLIST,XREF
```

PAUSE — Await Manual Intervention

You may code PAUSE as either a command in your $JOBUTIL stream or by typing in PAUSE. as a command from your terminal keyboard. JUMP and LABEL commands are not permitted.

When a PAUSE command is read, $JOBUTIL prompts you for input. You communicate with the processor using the attention key and three PAUSE subcommands.

$JOBUTIL - Job Stream Processor *(continued)*

PAUSE subcommands are:

- ABORT - To end $JOBUTIL processing
- ENTER - To enter $JOBUTIL commands
- GO - To end the PAUSE mode and read the next command in the procedure data set.

Syntax:

```
PAUSE

Required: none
```

Operands ***Description***

None

Examples: Data set with PAUSE.

```
JOB      PAYROLL
LOG      ON
PROGRAM  WAGES
EXEC
REMARK   LAST DAY OF MONTH?
REMARK   RUN  MONTHEND
PAUSE
EOJ
```

Example: In PAUSE mode (terminal output).

```
PAUSE - * - ATTN: GO/ENTER/ABORT

PAUSE
> ENTER

ENTER COMMAND  PROGRAM
ENTER OPERAND  MONTHEND
ENTER COMMAND  EXEC
ENTER OPERAND
```

$JOBUTIL - Job Stream Processor *(continued)*

PROC — Execute Procedure

Use the PROC command to pass control to another procedure data set. The operand identifies the data set. A PROC command is not allowed in the called procedure data set. You cannot place PROC between a PROGRAM command and an EXEC command. The completion code of the last program executed in the subprocedure will be available for testing by the main procedure.

Syntax:

```
PROC     procedure-name,volume

Required: procedure-name
Default: volume is IPL volume
```

Operands	***Description***
procedure-name	Defines the procedure data set to which control is to be passed
volume	Defines the volume containing the procedure data set to which control is to be passed.

Examples:

```
PROC        PAYROLL,EDX002
PROC        PAYROLL
```

$JOBUTIL - Job Stream Processor *(continued)*

PROGRAM — Identify Program

Use the PROGRAM command to identify the program to be executed.

Syntax:

```
PROGRAM   program-name,volume

Required: program-name
Default: volume is IPL volume
```

Operands	***Description***
program-name	Defines the name of the program member to be executed.
volume	Defines the name of the volume containing the program member to be executed.

Examples:

```
PROGRAM    $DEBUG,EDX002
PROGRAM    CHECKS
```

$JOBUTIL - Job Stream Processor *(continued)*

REMARK — Display Remark

Use the REMARK command to output a message to the operator with log ON or OFF. The operand contains the message. The maximum message length is 62 characters.

Syntax:

```
 REMARK comment

Required: none
```

Operands	*Description*
comment	Defines the comment to be displayed.

By including an @ sign in column 9, the screen or printer is advanced one line before remark is displayed/printed.

Example:

```
REMARK @INSERT DISKETTE EDX005
```

$JOBUTIL - Job Stream Processor *(continued)*

* — Comment

Use an asterisk (*) in position 1 to indicate internal comments in the procedure data set. Comments can start in position 2. The internal comment is not printed.

Syntax:

```
  * internal comment

Required: none
```

Operands ***Description***

None

Example:

```
* THIS PROCEDURE IS A TEST CASE
```

$JOBUTIL - Job Stream Processor *(continued)*

Batch Job Example

The following examples list three procedure data sets. The last two procedures shown are invoked by the first.

Example 1: List data set BATCH on EDX003.

```
JOB      BATCH
LOG      $SYSPRTR
REMARK   THIS JOB SHOULD RUN UNINTERRUPTED
PROGRAM  JUTEST1,EDX003
NOMSG
DS       LOAD1,EDX003
PARM     F0F4F1F7
EXEC
PROGRAM  JUTEST2,EDX003
EXEC
JUMP     JMP1,EQ,-10
PROGRAM  CALCDEMO,EDX002
EXEC
LABEL    JMP1
PROGRAM  JUTEST3,EDX003
PARM     2
EXEC
JUMP     JMP2,LT,5
PROGRAM  $EDIT1N,EDX002
DS       EDITWORK,EDX002
EXEC
LABEL    JMP2
PROC     PROC1,EDX003
PROC     PROC2,EDX003
EOJ
```

Example 2: List data set PROC1 on EDX003.

```
LOG      ON
REMARK   PROC1
PROGRAM  JUTEST2,EDX003
EXEC
EOP
```

$JOBUTIL - Job Stream Processor *(continued)*

Example 3: List data set PROC2 on EDX003.

```
REMARK   PROC2
PROGRAM  JUTEST1,EDX003
DS       LOAD1,EDX003
PARM     ENDT
EXEC
EOP
```

$LOG - Log Errors into Data Set

$LOG records information concerning I/O errors onto a disk or diskette data set. You can display the information recorded on the log data set by using $DISKUT2.

To use $LOG you must include the SYSLOG module at system generation. If you want to record program check, soft exception, and machine check information, you must also include the CIRCBUFF module at system generation.

Log Data Set

Before the system can record I/O errors, you must allocate a log data set to contain the device and system information available at the time of the I/O error. The log data set should be at least eight records long (data type). The log data set may reside in a disk or diskette volume. The log data set contains one 256-byte log entry per record; the first two records are used for control information.

Invoking $LOG

$LOG can be invoked by EDL programs with the LOAD instruction issued from a terminal. The log data set must be defined by the DS= keyword on the LOAD statement. The only required parameter is the address of a 4-byte area that contains:

	Hexadecimal Displacement	Length (Bytes)
PARM 1	0	1
PARM 2	2	1

If PARM1="Y" , $LOG will terminate on log data set wrap. If PARM1= " " or "N" , a blank or null character, $LOG will not terminate when the log data set wraps.

If PARM2="Y" , error messages will be displayed on occurrence. If PARM2= " " or "N" , a blank or null character, error messages will not be displayed.

Coding Example

```
                 .
                 .
                 .
                LOAD    $LOG,PARMLIST,DS=(LOGDS,EDX002)
                 .
                 .
                 .
PARMLIST         EQU     *
                 DC      CL2'N'
                 DC      CL2'Y'
```

$LOG - Log Errors into Data Set *(continued)*

You may also invoke $LOG with the $L operator command, the $JOBUTIL utility, or with a LOAD instruction issued from a program coded in the Event Driven Language. You must load $LOG for any error logging to occur. If you are loading $LOG from an application program or by using a $JOBUTIL procedure, you must pass $LOG a 1-word parameter of zeroes.

Note: For the remote manager (RM1) to receive error log messages, you must also load either the host program (CJUALTHL) or the send program (CJUALTSL).

To activate error logging, load $LOG into any partition. $LOG prompts you for the name of the log data set.

```
 > $L $LOG
LOGDS (NAME,VOLUME):  LOGDS,EDX002
LOADING $LOG     23P,01:02:46, LP= 0000, PART=2
```

$LOG Commands

After you load $LOG and the system activates error logging, you can use the attention commands provided by $LOG to deactivate, reactivate, or terminate error logging. You can also reinitialize the log data set using one of these attention commands. All attention commands can be invoked from any terminal assigned to the same storage partition. Messages will always go to the terminal from which $LOG was actually loaded.

```
 > $L $LOG
 LOGDS (NAME,VOLUME):   LOGDS,EDX002
LOADING $LOG        23P,01:02:46, LP= 0000, PART= 2

*********************************************************
*      $ L O G    U T I L I T Y                         *
*                                                       *
*  THE FOLLOWING ATTENTION COMMANDS ARE AVAILABLE:      *
*    ATTN/$LOGOFF  - TEMPORARILY DEACTIVATE LOGGING     *
*    ATTN/$LOGON   - REACTIVATE LOGGING                 *
*    ATTN/$LOGINIT - INITIALIZE LOG DATA SET            *
*                    REACTIVATE LOGGING                 *
*    ATTN/$LOGTERM - TERMINATE LOGGING                  *
*    ATTN/$LOG     - REISSUE COMMAND LIST               *
*    ATTN/$LOGDISP - DISPLAY ERROR MSG ON OCCURRENCE    *
*    ATTN/$LOGTDW  - TERMINATE ON DATA SET WRAP         *
*                                                       *
*  WARNING:  DO NOT CANCEL ($C) THIS PROGRAM.           *
*********************************************************
  LOGGING ACTIVATED
```

Each command and its explanation is presented in alphabetical order on the following page. For an explanation of the $LOG output, refer to the *Problem Determination Guide*.

$LOG - Log Errors into Data Set *(continued)*

$LOG — Reissue Command List

Use $LOG to display the attention commands. You can also display the error messages from the utility itself that have nothing to do with the errors that $LOG is tracking.

$LOGDISP — Display Error Messages on Occurrence

Use $LOGDISP to display error messages as they occur. For example, if your log data set becomes full during error logging, $LOG immediately displays an error message. If you don't use $LOGDISP, you must use $LOG to display the errors.

$LOGINIT — Initialize Log Data Set/Reactivate Logging

Use $LOGINIT to clear the log data set. The system then writes a new log control record to indicate that the log data set contains no entries. $LOGINIT also restarts error logging.

$LOGOFF — Temporarily Deactivate Logging

Use $LOGOFF to suspend error logging ($LOG remains loaded but but will no longer log errors.)

$LOGON — Reactivate Logging

Use $LOGON to restart error logging.

$LOGTDW — Terminate on Data Set Wrap

Use $LOGTDW to end the $LOG utility if the log data set becomes full during error logging. If you don't use this command, the the system returns to the third record in the data set and begins writing over the existing entries.

Example: End $LOG when data set wrap occurs; display messages as they occur ($LOGDISP).

```
 > $L $LOG LOGDS,EDX002
 > $LOGTDW

$LOG WILL TERMINATE ON DATA SET WRAP
 > $LOGDISP

ERROR MESSAGES WILL BE DISPLAYED ON OCCURRENCE
```

$LOGTERM — Terminate Logging

Use $LOGTERM to end the error logging utility ($LOG is deactivated and removed from storage.)

For more information about I/O error logging, refer to the *Problem Determination Guide*. For information about system generation considerations, see the *Installation and System Generation Guide*.

$MEMDISK - Allocate Unmapped Storage as a Disk

Use $MEMDISK to allocate all or a portion of unmapped storage to use as a "disk." This disk resembles a single-volume diskette with the volume name MEMDSK. You must include the STORMGR module at system generation to be able to use unmapped storage and this utility.

Use the MEMDSK volume to create temporary work data sets for the$S1ASM assembler, the $EDXASM compiler, and $EDXLINK. You may also want to place the assembler itself and all the overlays on MEMDSK.. This will decrease assembly time.

Note: Since MEMDSK is part of the memory system, you will lose the volume in the event of a power failure. Use it only for work data sets, programs, and other files that you can recover if a power failure does occur.

Invoking $MEMDISK

You invoke $MEMDISK with the $L operator command.

```
 > $L $MEMDISK
LOADING $MEMDISK     27P,00:01:54, LP=2C00, PART=3

MEMDISK INITIALIZATION UTILITY
```

Once you load $MEMDISK, you can use MEMDSK with any of the the other utilities except $DASDI and $INITDSK.

$MEMDISK Commands

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

```
COMMAND (?):   ?

IV - LOAD MEMDSK SUPPORT AND INITIALIZE VOLUME
DV - DELETE MEMDSK FROM SYSTEM
SL - SET $LOADER TO LOAD FROM MEMDSK
RL - RESET $LOADER TO LOAD FROM DISK
SD - SET SYSTEM DEFAULT VOLUME TO MEMDSK
RD _ RESET SYSTEM DEFAULT VOLUME
EN - END UTILITY
```

Each command and its explanation is presented in alphabetical order on the following pages.

$MEMDISK - Allocate Unmapped Storage as a Disk *(continued)*

DV — Delete MEMDSK from the System

Use the DV command to delete MEMDSK from the system and release that part of unmapped storage.

Example: Delete the MEMDSK volume.

```
COMMAND(?):   DV

DELETE MEMDSK? (Y/N)   Y
MEMDSK DELETED

COMMAND(?):
```

EN — End $MEMDISK

Use the EN command to end the $MEMDISK utility.

Example: End $MEMDISK.

```
COMMAND(?):   EN
$MEMDISK ENDED AT 11:18:30
```

IV — Load MEMDSK Support and Initialize Disk

Use the IV command to allocate and initialize MEMDSK. The system then allocates all the resources it needs.

Example: Allocate and initialize MEMDSK volume.

```
COMMAND(?):   IV

SIZE IN RECORDS (0 TO EXIT):    1000
MAXIMUM NUMBER OF DATA SETS:    20

MEMDSK ALLOCATED AND INITIALIZED

COMMAND(?):
```

At IPL time, the system gives you the message UNMAPPED STORAGE=XXX (DEC) 2K BLOCKS where XXX will be the number in decimal of the available 2K blocks of unmapped storage. You can figure your available records by multiplying this number by 8. If you respond to SIZE IN RECORDS with a number that is larger than the unmapped storage area that is available, the system displays the following messages and then reprompt you.

```
INSUFFICIENT UNMAPPED STORAGE AVAILABLE
MAXIMUM AVAILABLE SIZE FOR MEMDSK = XXXX
```

$MEMDISK - Allocate Unmapped Storage as a Disk *(continued)*

After the system reprompts you for the SIZE IN RECORDS that you want, it prompts you for with MAXIMUM NUMBER OF DATA SETS. If you specify too many data sets, the system will display the following message and then reprompt you.

```
NUMBER OF DATA SETS MUST BE BETWEEN 1 - XXX
```

RL — Reset $LOADER to Load from Disk(ette)

Use the RL command to reset the loader table to its original contents.

Example: Reset $LOADER to load from disk.

```
COMMAND(?):   RL

$LOADER WILL NOW BE LOADED FROM DISK

COMMAND(?):
```

RD — Reset System Default Volume

Use the RD command to reset the system default volume to what it was before you issued the SD command.

Example: Reset system default to EDX002.

```
COMMAND(?):   RD

SYSTEM DEFAULT VOLUME SET TO EDX002

COMMAND(?):
```

SD — Set System Default Volume to MEMDSK

Use the SD command to set the default volume to MEMDSK.

Example: Set system default volume to MEMDSK.

```
COMMAND(?):   SD

SYSTEM DEFAULT VOLUME SET TO MEMDSK

COMMAND(?):
```

$MEMDISK - Allocate Unmapped Storage as a Disk *(continued)*

SL — Set $LOADER to Load from MEMDSK

Use the SL command to cause the system to load $LOADER from MEMDSK. This allows you to load programs much more rapidly since it acts as a resident loader. You must copy $LOADER to MEMDSK (with the $COPYUT1 utility) before you use this command. The system saves the contents of the loader table in case you wish to restore the table using the RL command.

Example: Set $LOADER to load from MEMDSK.

```
COMMAND(?):  SL

$LOADER WILL NOW BE LOADED FROM MEMDSK

COMMAND(?):
```

$MOVEVOL - Disk Volume Dump/Restore

With $MOVEVOL you can dump the contents of an Event Driven Executive direct access volume to diskette when such a volume spans several diskettes. You can also restore a volume from diskette to disk. Thus, $MOVEVOL makes it possible for you to transfer large amounts of data from one system to another or to create backup copies of an online data base.

Diskette Usage

Diskette Contents

The first of the set of diskettes used for the dump function, called the control diskette, records control information and the volume directory. The rest of the diskettes store the data portion of the volume you are dumping. Control information is recorded on each data diskette to identify the diskette contents and to ensure that it contains data related to the dump operation described on the control diskette.

Diskette Format

All diskettes must be formatted identically with $DASDI. You can use either single-sided or double-sided diskettes; however, all diskettes in a set must be the same type. Each diskette must contain a volume label in the standard format. The volume label must be a six-character field in which the last three characters are used for sequencing, for example, SAV000, SAV001, ..., SAVNNN, where nnn is the last diskette used. All diskettes used must have the same three-character prefix. The last three characters used for sequencing must start with 000 for the first diskette.

4966 Diskette Usage Considerations

If you are using the 4966 diskette magazine unit for your dump/restore operation, you can use diskette magazines or an individual diskette slot. If you use an individual diskette slot, then you must place all of the subsequent diskettes you mount in the same slot. If you use diskette magazines, you must have all of your diskettes in the correct sequence with no empty slots in the magazine. The first volume with the suffix 000 must be in slot number 1 of the first magazine. You can use either or both of the diskette magazines, A and B.

Invoking $MOVEVOL

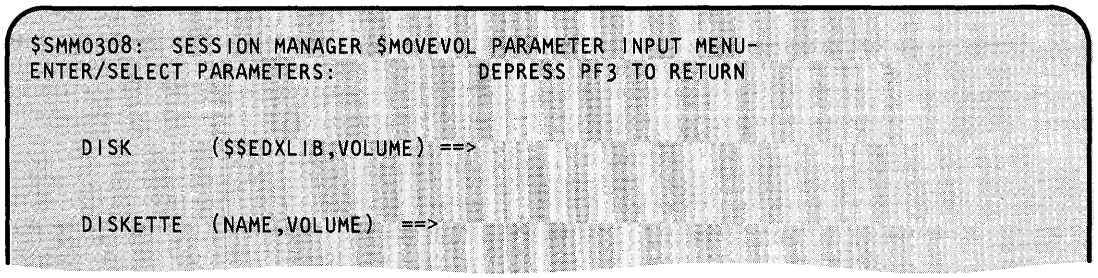
You invoke $MOVEVOL with the $L operator command or option 3.8 of the session manager.

$MOVEVOL - Disk Volume Dump/Restore *(continued)*

Data Set Specification

If you invoke $MOVEVOL with the $L operator command, the system prompts you to enter the names of the data sets and volumes to be used.

The following example shows the parameter menu displayed when you invoke $MOVEVOL with the session manager. Enter the requested information and press the enter key.



```
$SMM0308:  SESSION MANAGER $MOVEVOL PARAMETER INPUT MENU-
ENTER/SELECT PARAMETERS:           DEPRESS PF3 TO RETURN


    DISK      ($$EDXLIB,VOLUME) ==>


    DISKETTE  (NAME,VOLUME)  ==>
```

Figure 25. $MOVEVOL parameter input menu

Dump Procedure

The following steps are required to dump the contents of a direct access volume onto diskette.

1. Set up a control diskette.

   a. Use $INITDSK to:

      1) Initialize the control diskette with a volume label that is suffixed with 000 (for example, SAV000).

      2) Create a directory containing at least 1 member.

      3) If the diskette will be used to IPL another system, reserve space for a nucleus of the appropriate size and write the IPL text.

   b. Use $DISKUT1 to:

      1) Determine the directory size, in records, of the volume to be dumped.

      2) Change volume to the control diskette (for example, SAV000) and allocate a control data set with the same name as the name of the volume to be dumped. The member size of the control data set must be one record larger than the size of the directory of the volume being dumped.

$MOVEVOL - Disk Volume Dump/Restore *(continued)*

c. Use COPYUT1 to:

1) Copy other data sets onto the control diskette. For example, you may require $EDXNUC, the transient loader, or a copy of $MOVEVOL.

Note: The first record in the control data set contains control information and up to 50 characters of text describing the data being dumped. The remaining space stores a copy of the directory of the volume being dumped.

2. Set up a series of data diskettes. For each data diskette:

a. Use $INITDSK to:

1) Create a volume label. The volume label of each diskette must have the same three-character prefix as the control diskette and a three-character suffix indicating the sequence number, for example, SAV000, SAV001, ..., SAVNNN.

2) Create an owner ID field.

3) Allocate a one-member directory (the minimum number of members contained in a directory is one).

b. Use $DISKUT1 to:

1) Allocate a one-record data set named $CONTROL.

2) Allocate a second data set with the name of the volume to be dumped (for example, EDX002). The second data set must use the remaining available space on the diskette as follows:

Diskette type	Number of records at 128 BPS	Number of records at 256 BPS	Number of records at 1024 BPS
Diskette 1	946	1107	-
Diskette 2	1921	2217	-
Diskette 2-D	-	3845	4733

3. Mount the control diskette and load $MOVEVOL for execution.

You must specify two data sets at load time:

DISK The volume on disk to be dumped. Specify $$EDXLIB,volname.

DISKETTE The control data set on diskette. Specify dsname,volname.

$MOVEVOL - Disk Volume Dump/Restore *(continued)*

$MOVEVOL asks if you wish to dump from disk to diskette.

MOVEVOL then determines the number of additional diskettes required to dump the referenced volume (DISK), informs you of this requirement, and asks if the procedure should be continued. Reply N and the operation ends. Reply Y and the control information and disk directory are recorded on the control diskette and you are asked to mount a new diskette for transfer of the data portion of the volume being dumped.

4. Each time a diskette is filled, $MOVEVOL requests another diskette. Mount as many data diskettes as the system requests.

Example: Dump operation using a 4966 diskette magazine unit.

```
> $L $MOVEVOL
 DISK(NAME,VOLUME):   $$EDXLIB,EDX003
 DISKETTE(NAME,VOLUME):   $EDX003,SAV000

LOADING $MOVEVOL      20P,10:07:52, LP=5200, PART=2

DUMP VOLUME EDX003 FROM DISK TO DISKETTE?   Y

PROCESSING DISKETTE VOLUME SAV000
ENTER VOLUME IDENTIFICATION (1-50 CHAR.):  09/14/82
09/14/82 DUMP OF EDX003 - DATE IS 09/14/82
     11 MORE DISKETTE(S) REQUIRED, CONTINUE?  Y

PROCESSING DISKETTE VOLUME SAV000
PROCESSING DISKETTE VOLUME SAV001
PROCESSING DISKETTE VOLUME SAV002
PROCESSING DISKETTE VOLUME SAV003
PROCESSING DISKETTE VOLUME SAV004
PROCESSING DISKETTE VOLUME SAV005
PROCESSING DISKETTE VOLUME SAV006
PROCESSING DISKETTE VOLUME SAV007
PROCESSING DISKETTE VOLUME SAV008

MOUNT NEXT DISKETTE
REPLY -Y- WHEN DONE:   Y

PROCESSING DISKETTE VOLUME SAV009
PROCESSING DISKETTE VOLUME SAV010

VOLUME DUMP OPERATION COMPLETE
           9200 RECORDS TRANSFERRED

$MOVEVOL ENDED 10:10:13
```

$MOVEVOL - Disk Volume Dump/Restore *(continued)*

You may enter any text in response to the ENTER VOLUME IDENTIFICATION (1-50 CHAR.): prompt when dumping a volume. During the restore procedure, the information you entered here is redisplayed.

Restoration Procedure

The following steps are required for a restore operation.

1. Mount the control diskette and load $MOVEVOL for execution.
   a. Respond as described previously in dump procedure to requests for data sets.
   b. Select the restoration mode by responding N to the query for disk to diskette dump and Y to the query for diskette to disk restoration.
2. Mount data diskettes as requested.

Example 1: Restore operation using an individual 4966 diskette magazine slot. The source and target volumes are equal in size.

```
 > $L $MOVEVOL $$EDXLIB,EDX003 $EDX003,SAV000

LOADING $MOVEVOL      20P,11:05:05, LP=6300, PART=2

DUMP VOLUME EDX003 FROM DISK TO DISKETTE?  N

RESTORE VOLUME FROM DISKETTE TO DISK VOLUME EDX003?  Y
RESTORING DUMP OF EDX003 - DATE IS 09/14/82
CONTINUE?  Y

PROCESSING DISKETTE VOLUME SAV000
MOUNT NEXT DISKETTE
REPLY -Y- WHEN DONE:  Y

PROCESSING DISKETTE VOLUME SAV001
MOUNT NEXT DISKETTE
REPLY -Y- WHEN DONE:  Y

PROCESSING DISKETTE VOLUME SAV002
MOUNT NEXT DISKETTE
REPLY -Y- WHEN DONE:  Y

PROCESSING DISKETTE VOLUME SAV003
MOUNT NEXT DISKETTE
REPLY -Y- WHEN DONE:  Y

PROCESSING DISKETTE VOLUME SAV004

VOLUME INSTALLED
         3600 RECORDS TRANSFERRED

$MOVEVOL ENDED 11.10.56
```

$MOVEVOL - Disk Volume Dump/Restore *(continued)*

Example 2: Restore operation using a 4964 diskette unit. The source is smaller in size than the receiving volume.

```
> $L $MOVEVOL
 DISK    (NAME,VOLUME):  $$EDXLIB,MACLIB
 DISKETTE(NAME,VOLUME):  $MACLIB,BAC000
LOADING $MOVEVOL     20P,00:26:08, LP= 7000, PART=1
DUMP VOLUME MACLIB FROM DISK TO DISKETTE?  N

RESTORE VOLUME FROM DISKETTE TO DISK VOLUME MACLIB?  Y
RESTORING 5719-XX5 V04M00 9/12/82 TO VOLUME MACLIB
CONTINUE?  Y
SOURCE VOLUME IS SMALLER THAN THE TARGET.  CONTINUE?  Y
COMPRESS THIS VOLUME AFTER INSTALLATION.
PROCESSING DISKETTE VOLUME BAC000
MOUNT NEXT DISKETTE
REPLY -Y- WHEN DONE:  Y
PROCESSING DISKETTE VOLUME BAC001
   .
   .
   .
```

$MSGUT1 - Message Utility

$MSGUT1 formats source messages into a form suitable for use with the message handler. Once you have created a source-message data set, $MSGUT1 takes the source messages, converts them to either disk or storage-resident format, and saves them in another data set which you specify. If you have not allocated this data set previously using $DISKUT1, $MSGUT1 allocates it for you. For information on creating a source-message data set, see the *Event Driven Executive Language Programming Guide*.

Invoking $MSGUT1

You invoke $MSGUT1 with the $L operator command or option 2.14 of the session manager. After you invoke $MSGUT1, it prompts you for a work data set. This work data set must be at least as large as the source-message data set.

```
> $L $MSGUT1
WORKFILE (NAME,VOLUME):
```

$MSGUT1 Commands

To display the $MSGUT1 commands at your terminal, enter a question mark in response to the prompting message COMMAND (?):.

```
COMMAND (?):  ?

  DSK   - CONVERT SOURCE TO DISK-RESIDENT FORMAT
  LST * - DIRECT OUTPUT TO A TERMINAL
  PRT   - PRINT MESSAGES
  STG   - CONVERT SOURCE TO STORAGE-RESIDENT FORMAT
  END   - END THE MESSAGE UTILITY

COMMAND (?):
```

After $MSGUT1 displays the commands, it prompts you with COMMAND (?): again. Then you can respond with the command of your choice (for example, PRT).

Each command and its explanation is presented in alphabetical order on the following pages.

$MSGUT1 - Message Utility *(continued)*

DSK — Convert Source to Disk-Resident Format

Use the DSK command to convert source messages to the format used for disk-resident messages. The DSK command converts a source-message data set to disk-resident format and stores the messages in a data set on disk or diskette. Messages converted to disk-resident format do not contain variable information or comments. The variable information and comments are stripped off, control bytes are inserted at the beginning of each message, and the 256-byte EDX records are restructured into four logical records of 64 bytes for each message.

DSK prompts you for the name of the source-message data set and the volume on which it resides. After you enter this information, DSK prompts for the name of the disk-resident data set and the volume where the messages, once converted, are to be stored.

Example: Convert a source-message data set (USRSRC) on USRVOL and store the messages in a disk-resident data set (USRMSG) on the USRVOL.

```
COMMAND (?):  DSK
MESSAGE SOURCE DATA SET (NAME,VOLUME):  USRSRC,USRVOL
DISK-RESIDENT DATA SET (NAME,VOLUME):  USRMSG,USRVOL

START OF DISK MESSAGE PROCESSING BEGINS

DISK-RESIDENT MESSAGES STORED IN USRMSG,USRVOL

COMMAND (?):
```

END — End $MSGUT1

Use the END command to end $MSGUT1.

Example:

```
COMMAND (?):  END

$MSGUT1 ENDED AT 01:36:04
```

$MSGUT1 - Message Utility *(continued)*

LST — Direct Output to a Terminal

Use the LST command to direct output to a device other than the $SYSPRTR. Use the LST command with the PRT, STG, and DSK commands. Unless you change the output device with the LST command, all listings requested by the PRT, STG, and DSK commands are directed to the $SYSPRTR. If you specify LST *, the listing appears on the terminal where you invoked $MSGUT1.

Note: When you use the PRT, STG, and DSK options, all error messages are directed to the $SYSPRTR unless you specify otherwise. If any errors are detected, the following message is displayed:

```
FUNCTION HAS COMPLETED WITH ERRORS.
CHECK ALL OUTPUT.
```

Example: A listing of the messages in EDITUSER on USRVOL is first directed to the MPRINTER and then to the terminal labeled TERM01.

```
COMMAND (?):  LST MPRINTER

COMMAND (?):  PRT

MESSAGE SOURCE DATA SET (NAME,VOLUME):  EDITUSER,USRVOL
INCLUDE MSGID (Y/N)?:  Y
ENTER 4-CHARACTER MESSAGE
NUMBER PREFIX (DEFAULT MSG#):  PGMA

MESSAGE SOURCE DATA SET PROCESSING BEGINS

MESSAGES PRINTED ON MPRINTER

COMMAND (?):  LST TERM01

COMMAND (?):  PRT

MESSAGE SOURCE DATA SET (NAME,VOLUME):  EDITUSER,USRVOL
INCLUDE MSGID (Y/N)?:  Y
ENTER 4 CHARACTER MESSAGE
NUMBER PREFIX (DEFAULT MSG#):  PGMA

MESSAGE SOURCE DATA SET PROCESSING BEGINS

MESSAGES PRINTED ON TERM01

COMMAND (?):
```

$MSGUT1 - Message Utility *(continued)*

PRT — Print Messages

Use the PRT command to obtain a hard-copy listing of the messages in a specific source-message data set. You have the option of requesting a listing of the messages in the source-message data set with or without MSGIDS (consisting of the four-character component IDand the message number). The listing is directed to the $SYSPRTR. If you want to direct the listing to another output device name, use the LST command.

Example: Request a listing of the messages and MSGIDS in EDITUSER on USRVOL.

```
COMMAND (?):  PRT

MESSAGE SOURCE DATA SET (NAME,VOLUME):  EDITUSER,USRVOL
INCLUDE MSGID (Y/N):  Y
ENTER 4 CHARACTER MESSAGE
NUMBER PREFIX (DEFAULT MSG#):  PGMA

MESSAGE SOURCE DATA SET PROCESSING HAS BEGUN

COMMAND (?):
```

Following is a sample of the listing that appears on $SYSPRTR.

```
PGMA001 @SPOOL ACTIVE LIMIT REACHED
PGMA002 @SPOOL JOB CAPACITY REACHED
PGMA003 @SPOOL DATA SET FULL
PGMA004 ENTER COMMAND:
PGMA005 @DALL COMMAND COMPLETED
PGMA006 GENERIC JOB NAME PREFIX:
PGMA007 @DG COMMAND COMPLETED@
PGMA008 @SPOOL COMMAND INVALID@
PGMA009 @SPOOL COMMAND REJECTED - STOP PENDING@
```

$MSGUT1 - Message Utility *(continued)*

STG — Convert Source-Message to Storage-Resident Format

Use the STG command to convert a source-message data set to storage-resident format (object-like) for systems that do not have a disk or diskette or for better performance. The converted data set is stored in a disk data set that must be linked with application programs. The name of the disk data set containing the converted data must be the same as the name specified in the COMP statement. For a description of the COMP statement, see the *Language Reference*. This data set includes a cross-reference table with the address of the beginning of each message number. The address of the cross-reference table is an entry point in the storage-resident module and is used by the supervisor to access these messages.

STG prompts for the source-message data set and the name and volume where the storage-resident message module is to be saved.

Example: Store a source-message data set (MSG1SRC on USRVOL) in a data set (MSG1OBJ) on volume OBJLIB in main storage.

```
COMMAND (?):  STG
MESSAGE SOURCE DATA SET (NAME,VOLUME):   MSG1SRC,USRVOL
STORAGE-RESIDENT MODULE (NAME,VOLUME):   MSG1OBJ,OBJLIB

START OF STORAGE MESSAGE PROCESSING

STORAGE-RESIDENT MODULE STORED IN MSG1OBJ,OBJLIB

COMMAND (?):
```

$PFMAP - Identify 3101(Block Mode)/4978/4979/4980 Program Function Keys

The $PFMAP utility identifies the program function keys on the 3101 (Block Mode), 4978, 4979 and 4980 display stations. As you press each key, $PFMAP displays the associated system code in decimal and hexadecimal. A key's associated system code is the identification returned at completion of a WAIT KEY instruction or an ATTNLIST interrupt. The hard-copy key is active during execution of this program, and the system does not display its code. Press the enter key to end $PFMAP.

Invoking $PFMAP

You invoke $PFMAP with the $L operator command or option 4.6 of the session manager.

After you load $PFMAP, it displays two columns, DECIMAL and HEX. For each PF key you press, $PFMAP displays the system code in decimal and hexadecimal.

Example:

```
> $L $PFMAP
LOADING $PFMAP    2P,00:52:11, LP= 9200, PART=1

DECIMAL     HEX
   2        02
   3        03
   5        05
(Press enter key to end $PFMAP)
$PFMAP ENDED AT 00:54:51
```

Note: The interrupt key system code displayed in the $PFMAP utility differs from code listed in the hardware manual. Coding is always reduced by the value specified for PF1 at system generation. If you used the default value for the PF1 key at system generation, it decreases by one, the default value. If another number was specified as the value for PF1 at system generation, it will decrease by that number.

$PREFIND - Prefinding Data Sets and EDL Overlays

The $PREFIND utility locates the disk, diskette, and tape data sets and EDL overlay programs referenced by your program and stores their addresses in the header of your program. After $PREFIND has executed, program load time is shortened because the tasks performed by EDX are reduced.

$PREFIND is most effective when it is used to process programs that reference a large number of disk, diskette, or tape data sets and EDL overlay programs and when these programs must be loaded frequently from disk or diskette. We recommend using $PREFIND if your operating environment is relatively "static" or unchanging.

Program Load Process Overview

If a program uses data sets or EDL overlays programs (DS= and PGMS= parameters in PROGRAM statement), the assembler creates control blocks in the program header for each data set and EDL overlay program specified. These control blocks contain space for the physical addresses of the data sets and EDL overlay programs defined.

All data sets and EDL overlay programs required by a program are located and their sizes determined each time the program is loaded for execution. Thus, the loaded program executes correctly even if the size or location of one or more of the data sets or EDL overlay programs it uses has changed.

After you complete the program preparation process with $EDXLINK or $UPDATE, you can load the executable load module to storage.

When a large number of data sets and/or EDL overlay programs are defined, loading can be a time-consuming process because EDX must search a volume directory for each data set and program used. Thus, in a relatively "static" or unchanging environment, such as a production application, $PREFIND is useful in reducing the run time of the application.

$PREFIND allows data sets and EDL overlay programs to be located prior to program load time and writes physical addresses directly into the program header on disk or diskette. When the program is loaded, the information required is already present, so load time is reduced.

Invoking $PREFIND

You invoke $PREFIND with the $L operator command, option 2.11 of the session manager, or the job stream processor ($JOBUTIL).

Prefinding Data Sets and Overlays Using the $L Command

When you invoke $PREFIND with the $L operator command, it prompts you for the information it requires.

```
> $L $PREFIND
LOADING $PREFIND      27P,00:06:15, LP= 9800, PART=2
COMMAND (?):
```

$PREFIND - Prefinding Data Sets and EDL Overlays *(continued)*

$PREFIND Commands

To display the commands available under $PREFIND, enter a question mark (?) in response the COMMAND (?): prompt.

```
COMMAND (?):  ?

  PF - PRELOCATE DATA SETS AND OVERLAYS
  DE - DELETE PREFOUND STATUS
  EN - END THE PROGRAM

COMMAND (?):
```

After $PREFIND displays the commands, it prompts you again with COMMAND (?):. Enter the command of your choice.

If you enter either PF or DE, $PREFIND prompts you for the name and volume of the program and the numbers (1 through 9) of the data sets and EDL overlay programs that are to be located or whose prefound status is to be deleted.

You can enter all of the required information without waiting for the prompting messages. For example:

```
COMMAND (?):  PF MYPROG,EDX003 D=(1,2,4,7) P=(1,2,3)
```

The numbers in parentheses correspond to the numbers used in the DSN and PGMN parameters on the READ, WRITE, and LOAD Event Driven Language instructions.

You must always enter the data set and program numbers in the formats D=(__,__,__) and P=(__,__). Data set numbers, if present, must always precede EDL overlay program numbers. The word ALL can be used in place of the number string within the parenthesis. If you make a null response to the prompt for either (but not both) the data set or program numbers, no change in the status of the data sets or programs will occur. If you enter a number larger than the number of the largest data set/program, this information will be ignored and not cause an error. In other words, if you have two data sets listed in the program header, and you enter D=(1,3,5) for the data set numbers' prompt, the first data set will be prefound. The 3 and 5 will be ignored.

Enter only those data set or program numbers whose status is to be changed. For example, if a program references six data sets and you desire to prelocate the first three and the sixth, enter D=(1,2,3,6) when using the PF command. If at a later time you desire to delete the third data set from the prefound state, use a DE command specifying D=(3).

$PREFIND - Prefinding Data Sets and EDL Overlays *(continued)*

After performing the PF and DE commands described above, data sets 1, 2, and 6 are prefound and data sets 3, 4, and 5 are not. The execution of the DE command only affects DS3 and does not update the information in the program header concerning DS1, DS2, or DS6.

In the PROGRAM instruction, you can specify "dsname,??" format. For example, you may have coded the PROGRAM instruction as follows:

```
TASK1     PROGRAM    START1,DS=(??,(NAME2,EDX002)),
```

In this statement, two data sets are defined. The name of DS1 will be specified at load time. When the program is loaded, you are prompted for the name of DS1.

During execution of the DE command, you can delete a data set specified in this manner. $PREFIND prompts you to enter the name of this data set to be placed back into the program header since the original name was overridden during the previous PF command. If you invoke the DE command via $JOBUTIL, an error message occurs if the above condition is encountered.

Any data set or EDL overlay program not marked as being in the prefound state is located by $LOADER whenever the program is loaded into storage for execution.

Example 1: Processing multiple programs with prompting messages.

```
COMMAND (?):  PF
PGM(NAME,VOLUME):  TESTPREF,EDX001
ENTER DATA SET NUMBERS:  D=(1,2,3,7,9)
ENTER OVERLAY PGM. NUMBERS:  P=(ALL)

COMMAND COMPLETED
COMMAND (?):
```

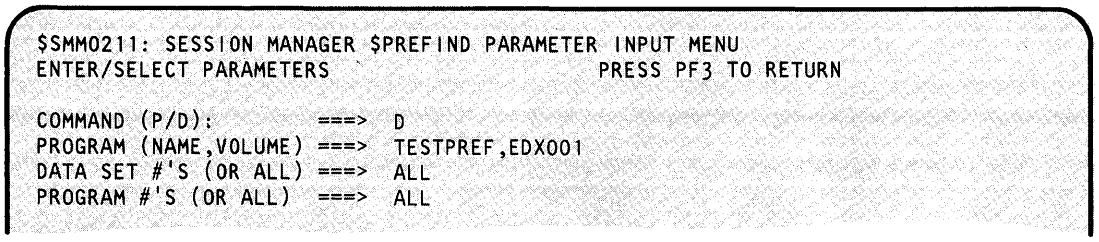
Example 2: Processing multiple programs without prompting messages.

```
COMMAND (?):  PF
PGM(NAME,VOLUME):  TESTPRE2,EDX001 D=(ALL) P=(ALL)

COMMAND COMPLETED
COMMAND (?):  EN

$PREFIND ENDED AT 00:10:59
```

$PREFIND - Prefinding Data Sets and EDL Overlays *(continued)*

Prefinding Data Sets and Overlays Using the Session Manager

You can use the session manager option menu for program preparation to invoke $PREFIND. Figure 26 shows the parameter input menu displayed when you invoke $PREFIND with the session manager.



```
$SMM0211: SESSION MANAGER $PREFIND PARAMETER INPUT MENU
ENTER/SELECT PARAMETERS                          PRESS PF3 TO RETURN

COMMAND (P/D):          ===>  D
PROGRAM (NAME,VOLUME) ===>  TESTPREF,EDX001
DATA SET #'S (OR ALL) ===>  ALL
PROGRAM #'S (OR ALL)  ===>  ALL
```

Figure 26. $PREFIND parameter input menu

After you enter the parameters, $PREFIND executes as if you had invoked it with $JOBUTIL.

Prefinding Data Sets and Overlays Using $JOBUTIL

When invoked through $JOBUTIL, $PREFIND requires the same information as described under "Prefinding Data Sets and Overlays Using the $L Command" on page UT-476. You provide this information with a PARM command having the format shown below. The number of spaces between the operands in the PARM command may be one or more, as long as the total number of characters, including spaces, does not exceed 62.

Syntax:

```
Character
Position
1       10
PARM    c progname,volume D=(__,__,__) P=(__,__,__)
```

Operands	*Description*
c	Either character P for PREFIND or character D for DELETE.
progname	The name of the program whose data set and EDL overlay program status is to be modified.
volume	The volume of the program whose data set and EDL overlay program status is to be modified.
__,__,__	The string of data set or EDL overlay program numbers, or the word ALL.

$PREFIND - Prefinding Data Sets and EDL Overlays *(continued)*

When invoked with $JOBUTIL, $PREFIND performs either a single prefind or delete function and then ends. $PREFIND directs error and/or termination messages to the device defined as $SYSPRTR.

Example: The following is an example of a $JOBUTIL procedure for prefinding data sets and EDL overlays.

```
LOG       $SYSPRTR
JOB       PREFIND
PROGRAM   $PREFIND,EDX001
NOMSG
PARM      D TESTPREF,EDX001 D=(ALL) P=(ALL)
EXEC
EOJ
```

$PRT2780 and $PRT3780

The $PRT2780 and $PRT3780 utilities print the spool records produced by the $RJE2780 and $RJE3780 utilities.

Invoking $PRT2780 and $PRT3780

You invoke $PRT2780 and $PRT3780 with the $L operator command or session manager options 8.6 and 8.7, respectively. When either utility is loaded, it prompts you for the name of the spool file to be printed. The utility ends when it reaches the end of the spool file. An initial option allows you to choose a printer other than $SYSPRTR if you want.

```
> $L $PRT3780
 DS1(NAME,VOLUME):  ASMWORK
LOADING $PRT3780     27P,00:02:44, LP= 8000, PART=2
PRINT TO $SYSPRTR? (Y OR N):  Y
$PRT3780 ENDED AT 00:03:05
```

Spooled data from a /*DR HASP command during a remote job entry session as printed out by the $PRT3780 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
```

$PRT2780 and $PRT3780 *(continued)*

Sample $RJE Session

Figure 27 shows a typical remote job entry session using the $RJE2780 and $PRT2780 utilities.

```
 > $L $RJE2780
LOADING $RJE2780     42P,00:00:00, LP= 7C00, PART=2

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):  RJEJOB01,EDX002
SUBMIT FILE READY TO SEND
FILE TRANSMISSION STARTED
FILE TRANSMISSION COMPLETED
 > COMMAND
ENTER COMMAND
/*$DA
COMMAND READY TO SEND
COMMAND SENT
 > PRINTON
ENTER PRINTER NAME:  PRTR1
PRTR1 DEFINED AS RJE PRINTER
 > COMMAND
ENTER COMMAND
/*$DA
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
```

Figure 27 (Part 1 of 2). Sample $RJE session

$PRT2780 and $PRT3780 *(continued)*

```
> SPOOL
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):  PCHOUT02,EDX002
PUNCH FILE DEFINED
PUNCHING STARTED
PUNCHING COMPLETED
LAST CARD PUNCHED WAS CARD 1 ON RECORD      51
> ENDSPOOL
SPOOLING COMPLETED
> COMMAND
ENTER COMMAND
/*SIGNOFF
COMMAND READY TO SEND
COMMAND SENT

$RJE2780 ENDED AT 00:00:00
> $L $PRT2780
 DS1(NAME,VOLUME):  SPOOL01,EDX002
LOADING $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 27 (Part 2 of 2). Sample $RJE session

$RJE2780 and $RJE3780

$RJE2780 and $RJE3780 are utilities that simulate an IBM 2780 and 3780, respectively. The term "RJE utility" refers to both $RJE2780 and $RJE3780.

The $RJE2780 utility simulates an IBM 2780 having the following characteristics and features:

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

The $RJE3780 utility simulates an IBM 3780 having the following characteristics and features:

- 3780 with IBM 3781 Card Punch
- Compression for both input and output
- Transparent or nontransparent (compressed) punch output.

Interface to Host RJE Subsystems

$RJE2780 and $RJE3780 present the same interface to the following System/360 and System/370 host RJE subsystems:

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

$RJE2780 and $RJE3780 *(continued)*

Invoking $RJE2780 or $RJE3780

You invoke $RJE2780 and $RJE3780 with the $L operator command or session manager options 8.4 and 8.5, respectively.

When the $RJE utility is first loaded, it checks for the presence of only one BSC line specified in the supervisor. If there is only one line, the actual device address of the adapter is used as the default line address and the utility issues no prompt. If more than one BSC line has been defined, the $RJE utility prompts you for the RJE line address. Subsequent control operations are all performed using the $RJE attention commands. Figure 28 lists these commands.

You can load multiple copies of $RJE using different lines to the host. You can use the spool facility to avoid contention for a single printer. Figure 27 on page UT-482 shows a sample $RJE session.

ABORT	Stops transmission to or from the host
COMMAND	Sends a single card image to the host
ENDRJE	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 28. $RJE attention commands

$RJE2780 and $RJE3780 *(continued)*

Attention Commands

Eleven commands are available to perform control operations.

ABORT

Use the ABORT command to stop a data transmission that 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, you should enter any desired commands (for example, submitting another job, cancelling the current output, holding a job, or displaying status) before you enter the ABORT command.

The ABORT 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

Use the COMMAND command to send a single card image record to the host. The most common use of this capability is to send control commands and information requests to the host. For example, this command should be used to terminate BSC job stream processing on JES2 and JES3 host systems by sending the signoff control statement, /*SIGNOFF.

After entering COMMAND, you are asked to enter the actual command to be sent to the host. For a list of valid commands and requests, refer to the operator reference manuals pertaining to your system.

ENDRJE

Use the ENDRJE command to end the $RJE utility program.

ENDSPOOL

Use the ENDSPOOL command to end the spooling of printer output (see SPOOL). If a print data stream is being received and spooled when you enter this command, spooling continues until the end of the data stream. Subsequent print data streams will then be printed on the defined printer.

PRINTON

Use the PRINTON command to define the name of the terminal to be used for print output. If you do not specify this, $SYSPRTR is assumed. If a print data stream is being received and printed when you enter this command, the print data stream continues to print on the same device until the end of the data stream is reached. Subsequent print data streams will then be printed on the newly defined printer.

$RJE2780 and $RJE3780 *(continued)*

PUNCHS and PUNCHO

Use the PUNCHS and PUNCHO commands to define the disk or diskette file to receive punch data from the host. Card image punch data streams are written to disk in two different formats: source (S) or object (O). Source format produces two 80-byte card image records per 256-byte disk record with the second card starting at byte location 129. Object format produces 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.

When you enter PUNCHS or PUNCHO, the $RJE utility prompts for the name and volume of the file to be used for punch output. If you do not specify volume, 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,EDX001
```

$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). You must modify $RJE to purge other than HASP punch header cards.

RESET

Use the RESET command to reset functions that have *not* started operation in $RJE (for example, buffered command images that have not yet been sent to the host and SUBMIT files that have not yet started transmission). Use RESET with caution: if you issue RESET while a function is in process or if you overlap a function initiation sequence, you may get unpredictable results. 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
```

$RJE2780 and $RJE3780 *(continued)*

SPOOL

Use the SPOOL command to define the disk or diskette file to receive printer data from the host. If you do not define a file, $RJE prints received data directly to the printer. Once specified, all printer output is spooled until you issue an ENDSPOOL command The $PRT2780 or $PRT3780 utility is used to print the contents of a spool file produced by $RJE2780 or $RJE3780, respectively.

Upon entering the SPOOL command, you are prompted for the name and volume of the disk or diskette file to be used for printer output. If you do not specify a volume, 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. You can enter the spool file name and volume with the SPOOL command, for example:

```
SPOOL SPOOLFLE,WRKLIB
```

SUBMIT and SUBMITX

Use the SUBMIT and SUBMITX commands to define and send, respectively, a data stream or transparent data stream to the host. You can send multiple disk or diskette files 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:

/*END, which signifies the end of the data stream to be sent; and

/*CONCAT filename,volume, which signifies that the data stream is to be continued using the file specified. If you do not specify a volume, the IPL volume is assumed. You can concatenate any number of files into one data stream, but the /*CONCAT statement can only be used once in each file. It should be the last statement since the utility uses the statement to open the next file and never returns to the previous or original file.

When you enter a SUBMIT or SUBMITX command, you are prompted for the name and volume of the file to be sent to the host. If you do not specify a volume, the IPL volume is assumed. You can also enter the submit file name and volume with the SUBMIT or SUBMITX command, for example:

```
SUBMITX MYJOB,WRKLIB
```

$SPLUT1 - Spool Utility

You can use $SPLUT1 to change the spooling capacity parameters and specify restart of the spooling facility under specific circumstances.

$SPLUT1 Operations

You can use the spool utility to perform the following operations:

1. Change any of the following spooling capacity parameters.

Command	**Function**
CD	Change the devices designated as spool devices and specify spool writer autostart.
DS	Change the spool data set to be used.
GS	Change the spool data set group size.
MA	Change the maximum number of active spool jobs.
MJ	Change the total number of spool jobs allowable in the system at any point in time.
SO	Change the separator page option.

   Figure 29 on page UT-491 lists the default parameters shipped with the spooling facility.

2. Specify that spooling is to be restarted using a specified, previously used spool data set.

   Use spool RS command to:

   - Restart spooling to retrieve the output spooled up to the last checkpoint in the event of a system failure. Checkpoint data is written to the spool data set each time a group of records is allocated to a spool job, each time a spool job becomes ready or is deleted, and each time a $S operator command is processed. For an explanation of the $S operator command, see "$S - Control Spooled Program Output" on page UT-19.

   - Restart spooling as a normal procedure after invoking the spool facility - whether or not any previously spooled data remains in the spool data set used for the restart.

   - Copy a spool data set to diskette and transfer to another machine and thus make spooled output available across systems.

   Note: Once the spool restart is specified, restart remains in effect for all subsequent spool sessions until reset with $SPLUT1.

$SPLUT1 - Spool Utility *(continued)*

3. Use the RS command to specify that spooling is to be cold-started, initializing the specified spool data set. A cold start should be specified:

   - The first time a new spool data set is used;
   - To delete any jobs left over from a previous spool session.

   Set the restart mode on (Y) to start spooling using a previously used spool data set. Use restart mode to:

   - Ensure that any data left from a previous session is not lost.
   - Restart spooling to retrieve the output spooling up to the last checkpoint before a system failure. Checkpoint data is written to the spool data set each time a group of records is allocated to a spool job, each time a spool job becomes ready or is deleted, and each time a $S operating command is processed.
   - Transfer a spool data set on diskette from another Series/1.

     Notes:

     a. The specified restart mode (Y/N) remains in effect for all subsequent spool sessions.

     b. Restart mode must be set to off (N) before you change any spooling capacity parameters.

     c. You cannot set restart mode to on (Y) during the $SPLUT1 session in which you change any spooling capacity parameters (CD, DS, GS, MA, MJ, and SO commands).

     d. The changes take effect in the next spooling session.

   - Specify the termination of $SPLUT1.

CA	Cancel the utility and ignore the changes.
EN	End the utility and record the changes.

$SPLUT1 - Spool Utility *(continued)*

Notes:

1. Items 1 and 2 are mutually exclusive in a single utility session. On any single invocation of the spool utility, you can do one or the other of these, but not both. That is, if you change the spooling support parameters, you cannot specify a spool restart; if you specify a spool restart, you cannot change the spooling capacity parameters.

2. Restart mode must be reset before you can use any of the spool capacity change parameters.

3. The changes do not affect the current spooling session. They take effect in the next spooling session.

4. With the shipped system, the $SYSPRTR writer is automatically started at the start of a spool session.

PARAMETER	DEFAULT
Spool data set name (DSNAME)	SPOOL
Spool data set volume (VOLUME)	EDX003
Maximum allowable spool jobs (MAXJOBS)	10
Maximum active spool jobs (MAXACTV)	4
Restart (RESTRT)	N
Spool data set group size (GRPSZ)	100
Spool data set groups (GRPS)	10
Separator page option (SEP)	Y
Designated spool devices (DEVICES)	$SYSPRTR

Figure 29. Spooling Defaults

Invoking $SPLUT1

You invoke $SPLUT1 with the $L operator command or option 4.7 of the session manager. You can load $SPLUT1 into any partition whether or not spooling is active.

```
> $L $SPLUT1
LOADING $SPLUT1      30P,01:00:16, LP= 0000, PART=1

*****************************************************
**                 EDX SPOOL UTILITY               **
** CHANGES EFFECTIVE THE NEXT TIME $SPOOL IS LOADED **
*****************************************************

DSNAME---VOLUME-MAXJOBS-MAXACTV-RESTRT-GRPS-GRPSZ-SEP--DEVICES-AUTO-FORM

SPOOL    EDX003    10       4      N       10   100    Y   $SYSPRTR Y

COMMAND(?):
```

$SPLUT1 - Spool Utility *(continued)*

$SPLUT1 Commands

To display the $SPLUT1 commands, enter a question mark in response to the COMMAND (?): prompt.

```
COMMAND(?):  ?

DS - CHANGE DATA SET NAME
MJ - CHANGE MAX JOBS
MA - CHANGE MAX ACTIVE JOBS
RS - RESTART
CD - CHANGE SPOOL DEVICE(S) (10 MAXIMUM)
GS - CHANGE SPOOL DATA SET GROUPSIZE
SO - SET SEPARATOR PAGE OPTION
CA - CANCEL UTILITY (CHANGES IGNORED)
EN - END UTILITY (CHANGES ACCEPTED)

COMMAND(?):
```

After $SPLUT1 displays the commands, it prompts you with COMMAND (?): again. Then you can respond with the command of your choice (for example, DS).

Spool Capacity Change Examples

Example - Change Spool Support Parameters

The following example shows how to:

- Change the spool data set to SPOOLTMP,EDX003.
- Increase the total number of spool jobs allowed in the system at any point in time to 14.
- Increase the maximum number of active spool jobs to 10.
- Designate the devices MPTR and LPTR as well as $SYSPRTR as spool devices.
- Specify automatic writer start for $SYSPRTR with forms code FM35.
- Specify no automatic writer start for LPTR and MPTR.
- Change the spool data set allocation (groupsize) to 20 records per group.
- Change separator option to no (do not write separator page between jobs).

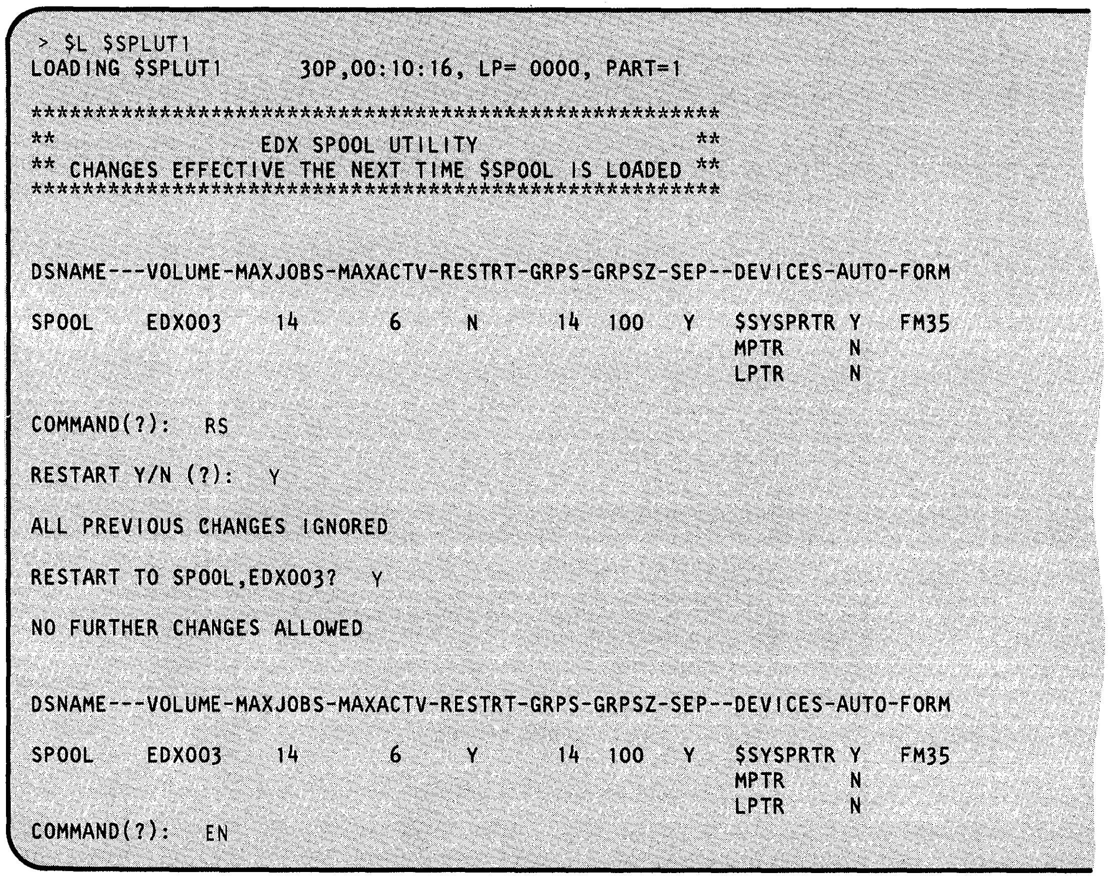
$SPLUT1 - Spool Utility *(continued)*

```
> $L $SPLUT1
LOADING $SPLUT1      30P,01:00:16, LP= 0000, PART=1

******************************************************
**                 EDX SPOOL UTILITY                **
** CHANGES EFFECTIVE THE NEXT TIME $SPOOL IS LOADED **
******************************************************


DSNAME---VOLUME-MAXJOBS-MAXACTV-RESTRT-GRPS-GRPSZ-SEP--DEVICES-AUTO-FORM

SPOOL    EDX003    10       4      N      14   100   Y   $SYSPRTR Y

COMMAND(?):  DS
ENTER(NAME,VOLUME):  SPOOLTMP,EDX003
DSNAME---VOLUME-MAXJOBS-MAXACTV-RESTRT-GRPS-GRPSZ-SEP--DEVICES-AUTO-FORM

SPOOLTMP EDX003    10       4      N      14   100   Y   $SYSPRTR Y

COMMAND(?):  MJ
MAX JOBS:  14
DSNAME---VOLUME-MAXJOBS-MAXACTV-RESTRT-GRPS-GRPSZ-SEP--DEVICES-AUTO-FORM

SPOOLTMP EDX003    14       4      N      14   100   Y   $SYSPRTR Y
COMMAND(?):  MA
MAX ACTIVE JOBS:  6
DSNAME---VOLUME-MAXJOBS-MAXACTV-RESTRT-GRPS-GRPSZ-SEP--DEVICES-AUTO-FORM
SPOOLTMP EDX003    14       6      N      14   100   Y   $SYSPRTR Y
COMMAND(?):  CD
DEVICE NAME (ENTER BLANK TO END):  $SYSPRTR
WRITER AUTOSTART ? (Y/N):  Y
ENTER FORMS CODE:  FM35
DEVICE NAME (ENTER BLANK TO END):  MPTR
WRITER AUTOSTART ? (Y/N):  N
DEVICE NAME (ENTER BLANK TO END):  LPTR
WRITER AUTOSTART ? (Y/N):  N
DEVICE NAME (ENTER BLANK TO END):
DSNAME---VOLUME-MAXJOBS-MAXACTV-RESTRT-GRPS-GRPSZ-SEP--DEVICES-AUTO-FORM

SPOOLTMP EDX003    14       6      N      14   100   Y   $SYSPRTR Y   FM35
                                                         MPTR     N
                                                         LPTR     N
COMMAND(?):  GS
PHYSICAL RECORDS PER GROUP (1-255):  20
DSNAME---VOLUME-MAXJOBS-MAXACTV-RESTRT-GRPS-GRPSZ-SEP--DEVICES-AUTO-FORM

SPOOLTMP EDX003    14       6      N      14    20   Y   $SYSPRTR Y   FM35
                                                         MPTR     N
                                                         LPTR     N
COMMAND(?):  SO
SET SEPARATOR PAGE OPTION (Y/N NOW IS Y):  N
DSNAME---VOLUME-MAXJOBS-MAXACTV-RESTRT-GRPS-GRPSZ-SEP--DEVICES-AUTO-FORM

SPOOLTMP EDX003    14       6 .    N      14    20   N   $SYSPRTR Y   FM35
                                                         MPTR     N
                                                         LPTR     N
COMMAND(?):  EN
```

$SPLUT1 - Spool Utility *(continued)*

Spool Start Type Examples

Example 1: Specify spool restart to current data set. This example shows how to specify that a spool restart is to be done using the same spool data set used in the last spooling session (SPOOL,EDX003).



```
 > $L $SPLUT1
LOADING $SPLUT1      30P,00:10:16, LP= 0000, PART=1

*****************************************************
**                EDX SPOOL UTILITY                **
** CHANGES EFFECTIVE THE NEXT TIME $SPOOL IS LOADED **
*****************************************************


DSNAME---VOLUME-MAXJOBS-MAXACTV-RESTRT-GRPS-GRPSZ-SEP--DEVICES-AUTO-FORM

SPOOL    EDX003    14       6      N       14  100   Y   $SYSPRTR Y   FM35
                                                          MPTR     N
                                                          LPTR     N

COMMAND(?):   RS

RESTART Y/N (?):   Y

ALL PREVIOUS CHANGES IGNORED

RESTART TO SPOOL,EDX003?   Y

NO FURTHER CHANGES ALLOWED

DSNAME---VOLUME-MAXJOBS-MAXACTV-RESTRT-GRPS-GRPSZ-SEP--DEVICES-AUTO-FORM

SPOOL    EDX003    14       6      Y       14  100   Y   $SYSPRTR Y   FM35
                                                          MPTR     N
                                                          LPTR     N
COMMAND(?):   EN
```

$SPLUT1 - Spool Utility *(continued)*

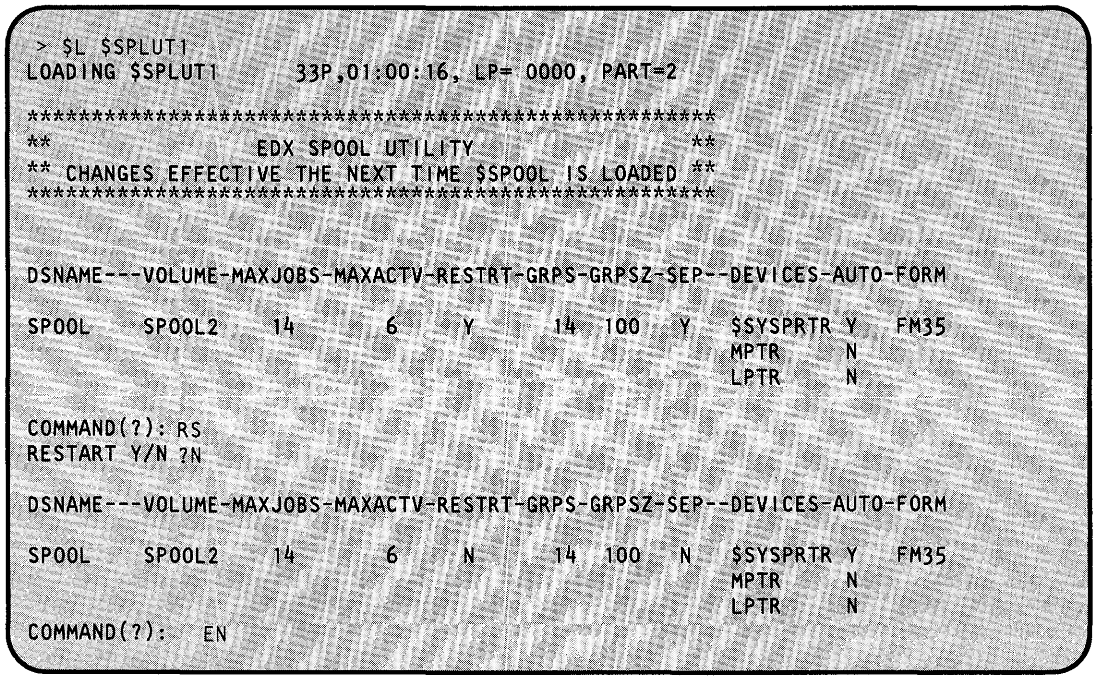
Example 2: Specify spool restart using data set SPOOL1. This example shows how to specify that a spool restart is to be done using a different spool data set than that used in the last spooling session.

```
> $L $SPLUT1
 LOADING $SPLUT1      30P,01:00:16, LP= 0000, PART=2

 ******************************************************
 **                EDX SPOOL UTILITY                 **
 ** CHANGES EFFECTIVE THE NEXT TIME $SPOOL IS LOADED **
 ******************************************************


 DSNAME---VOLUME-MAXJOBS-MAXACTV-RESTRT-GRPS-GRPSZ-SEP--DEVICES-AUTO-FORM

 SPOOL    EDX003    14       6      N      14  100    Y    $SYSPRTR Y    FM35
                                                           MPTR     N
                                                           LPTR     N

 COMMAND(?): RS

 RESTART Y/N (?): Y

 ALL PREVIOUS CHANGES IGNORED

 RESTART TO SPOOL,EDX003? N
 ENTER (NAME,VOLUME):  SPOOL1,EDX003

 NO FURTHER CHANGES ALLOWED


 DSNAME---VOLUME-MAXJOBS-MAXACTV-RESTRT-GRPS-GRPSZ-SEP--DEVICES-AUTO-FORM

 SPOOL1   EDX003    10       4      Y      15  100    Y    $SYSPRTR Y    FM35
 COMMAND(?): EN
```

Note: MAXJOBS, MAXACTV, GRPS, and DEVICES reflect the configuration of the spool data used in the last spooling session using that data set.

$SPLUT1 - Spool Utility *(continued)*

Example 3: Specify spool cold start. This example shows how to specify that spooling is to be cold-started.



```
 > $L $SPLUT1
LOADING $SPLUT1      33P,01:00:16, LP= 0000, PART=2

******************************************************
**                 EDX SPOOL UTILITY                **
** CHANGES EFFECTIVE THE NEXT TIME $SPOOL IS LOADED **
******************************************************


DSNAME---VOLUME-MAXJOBS-MAXACTV-RESTRT-GRPS-GRPSZ-SEP--DEVICES-AUTO-FORM

SPOOL    SPOOL2    14       6      Y      14  100   Y   $SYSPRTR Y   FM35
                                                        MPTR     N
                                                        LPTR     N

COMMAND(?): RS
RESTART Y/N ?N

DSNAME---VOLUME-MAXJOBS-MAXACTV-RESTRT-GRPS-GRPSZ-SEP--DEVICES-AUTO-FORM

SPOOL    SPOOL2    14       6      N      14  100   N   $SYSPRTR Y   FM35
                                                        MPTR     N
                                                        LPTR     N
COMMAND(?):  EN
```

$STGUT1 - Free Up Nonprogram Areas of Storage

$STGUT1 releases any areas of storage, within a specific partition, that are not assigned currently to application programs. It also prints out the segmentation registers of the partition you are in currently or the number of overlay segments in unmapped storage.

Invoking $STGUT1

Invoke $STGUT1 with the $L operator command. The session manager does not support this utility.

```
> $L $STGUT1
 $STGUT1 - FREE STORAGE UTILITY
 LOADING $STGUT1     9P,00:00:00, LP= 0000, PART= 2

 COMMAND (?):
```

$STGUT1 Commands

To display the $STGUT1 commands on your terminal, enter a question mark in response to the COMMAND (?): prompt.

```
COMMAND (?): ?

FR - FREE UP STORAGE
DS - LIST STORAGE SEGMENTATION REGISTERS
MX - MONITOR SYSTEM CONTROL BLOCKS
   (-CA- WILL CANCEL)
UN - LIST UNMAPPED STORAGE INFORMATION
EN - END PROGRAM
COMMAND (?):
```

After $STGUT1 displays the commands, you are prompted with COMMAND (?): again. Then you can respond with the command of your choice (for example, FR).

Each command and its explanation is presented in alphabetical order on the following pages.

$STGUT1 - Free Up Nonprogram Areas of Storage *(continued)*

DS — List Storage Segmentation Registers

Use the DS command to list the storage segmentation registers for the current partition. If you wish to list the segmentation registers of another partition, use the $CP operator command to change partitions.

Example: List segmentation registers.

```
COMMAND (?):   DS

BLOCK     ADS0 ADS1 ADS2 ADS3 ADS4 ADS5 ADS6 ADS7
 0000     0004 0104 0204 0304 0000 0000 0000 0000
 0800     000C 010C 020C 030C 0000 0000 0000 0000
 1000     0014 0114 0214 0314 0000 0000 0000 0000
 1800     001C 011C 021C 031C 0000 0000 0000 0000
 2000     0024 0124 0224 0324 0000 0000 0000 0000
   .
   .
   .
 D800     00DC 01DC 02DC 03DC 0000 0000 0000 0000
 E000     00E4 01E4 02E4 03E4 0000 0000 0000 0000
 E800     00EC 01EC 02EC 03EC 0000 0000 0000 0000
 F000     00F4 01F4 02F4 03F4 0000 0000 0000 0000
 F800     00FC 01FC 02FC 03FC 0000 0000 0000 0000

COMMAND (?):
```

EN — End Program

Use the EN command to end the $STGUT1 utility.

Example:

```
COMMAND (?):   EN
$STGUT1 ENDED AT 02:18:31
```

$STGUT1 - Free Up Nonprogram Areas of Storage *(continued)*

FR — Free Up Storage

Use the FR command to release storage that is not being used by application programs. $STGUT1 prompts you for the partition number and location (load point) of the storage you want to release. If the area is assigned to a program currently, $STGUT1 issues a message and does not release the area.

Example 1: Free up storage in partition 4 at load point 0.

```
COMMAND (?): FR
PARTITION = 4
LOAD POINT = 0

STORAGE RELEASED
```

Example 2: Attempt to free up storage beginning at load point 0 in partition 4 that is assigned to a program currently.

```
COMMAND (?): FR
PARTITION = 4
LOAD POINT = 0

PROGRAM AREA-NOT RELEASED
```

In this example, the storage area was assigned to a program or there was no storage to be released at the load point (address) you specified.

MX — Monitor System Control Blocks

Use the MX command to monitor the buffer used to track the number of interrupts that are received by the supervisor and the cross partition stack used by the supervisor when branching between partition one and supervisor code located in other partitions. The interrupt buffer and cross partition stack are defined with the SYSTEM definition station. The interrupt buffer IABUF is defined with the IABUF= operand and defaults to 20 interrupts. The cross partition stack XPSSTK is defined with the XPSSTK= operand and defaults to 20 entries. You can list the buffer and stack information on your terminal or write it out to disk or diskette.

To list the information on your terminal, use the > DI command under the MX command. $STGUT1 displays the most recent size of the IABUF and the XPSSTK, the maximum number of entries used to date (MAX USED ENTRIES), and the number of entries being used currently.

To write the information to a disk or diskette, $STGUT1 prompts you for a data set name and volume. If the specified data set has already been used to store buffer and stack information, $STGUT1 first reads the information contained in it and then automatically updates this information periodically. If the specified data set did not exist previously, STGUT1 creates and updates the information automatically. The data set is updated whenever the attention DI (> DI) command is used to display the information on the terminal. It is not necessary to enter the (>

$STGUT1 - Free Up Nonprogram Areas of Storage *(continued)*

DI) command after using the MX) command for the first time. Information appears automatically at that time only.

Use the MX command to obtain other types of information, such as SCBs and/or static and dynamic segmentation register information.

It is recommended that you check the IABUF and XPSSTK periodically to ensure that they are large enough for your system to continue executing. If the maximum used entries for either the IABUF or the XPSSTK is reaching the specified size, you should redefine the sizes on the SYSTEM definition statement and regenerate your system.

Example 1: List the XPS stack information on the terminal and cancel buffer and stack monitoring.

```
COMMAND (?):   MX

ATTN 'CA' CANCELS MONITORING
ATTN 'DI' LISTS SYSTEM INFORMATION ON TERMINAL
WRITE XPS STACK INFO TO DISK(ETTE)?   N

IABUF  SIZE  =  20;   MAX USED ENTRIES =   0;   CURRENT ENTRIES =    0
XPSSTK SIZE  =  20;   MAX USED ENTRIES =   2;   CURRENT ENTRIES =    1
 > CA

MONITORING CANCELLED

COMMAND (?):
```

Example 2: Write XPS stack information to diskette and display it on the terminal.

```
COMMAND (?):   MX

ATTN 'CA' CANCELS MONITORING
ATTN 'DI' LISTS SYSTEM INFORMATION ON TERMINAL
WRITE SYSTEM INFORMATION TO DISK(ETTE)?   Y

ENTER OUTPUT DATA SET (NAME,VOLUME):   STACK,EDX40

IABUF  SIZE  =  20;   MAX USED ENTRIES =   0;   CURRENT ENTRIES =    0
XPSSTK SIZE  =  20;   MAX USED ENTRIES =   2;   CURRENT ENTRIES =    1
 > DI
IABUF  SIZE  =  20;   MAX USED ENTRIES =   0;   CURRENT ENTRIES =    0
XPSSTK SIZE  =  20;   MAX USED ENTRIES =   2;   CURRENT ENTRIES =    1
 > CA

MONITORING CANCELLED

COMMAND (?):
```

$STGUT1 - Free Up Nonprogram Areas of Storage *(continued)*

UN — List Unmapped Storage Information

Use the UN command to list the number of overlay segments residing in unmapped storage for a specific program.

```
COMMAND (?):  UN
PROGRAM NAME:  EDXLINK
```

If the program you specify is not active currently, $STGUT1 issues the following message:

```
PROGRAM EDXLINK NOT FOUND
```

If the program specified does not require any overlay segments in unmapped storage or no overlay segments have been referenced, $STGUT1 issues the following message:

```
PROGRAM=  EDXLINK   PART=  2
NO STORAGE BLOCKS FOUND
```

If the program specified does require overlay segments in unmapped storage, $STGUT1 issues the following messages (an explanation of the numbered items follows the example):

```
1      COMMAND (?):  UN EDXLINK

       PROGRAM= EDXLINK   PART=  3

2      UNMAPPED STORAGE BLOCK #  1

3      OVERLAY MANAGER STORAGE BLOCK

4      NUMBER OF UNMAPPED STORAGE AREAS REQUESTED=   5

5      NUMBER OF STORAGE-RESIDENT AREAS OBTAINED=   5

6      ADDRESS OF MAPPED AREA=   6000   PART=   3

       COMMAND (?):
```

$STGUT1 - Free Up Nonprogram Areas of Storage *(continued)*

1 The name of the program requiring overlay segments in unmapped storage.

2 The number of the storage block that is being described. (If more than one control block is described, information about each control is displayed.)

3 The overlay manager has requested the storage block area. The overlay manager is automatically invoked by the $EDXLINK UNMAPCNT control statement.

4 The number of unmapped storage areas requested.

5 The number of unmapped storage areas actually obtained. If the number is less than requested, the program will still execute with the areas not placed in unmapped storage being referenced off the disk.

6 The address and partition of the mapped area.

$SUBMIT - Submit/Control Jobs in Job Queue Processor

The $SUBMIT utility performs several functions to submit jobs ($JOBUTIL command data sets) and control processing of those jobs by the job queue processor. $SUBMIT enables you to:

- Submit a job for execution.
- Submit and hold a job.
- Hold a job waiting for execution.
- Release a held job for execution.
- Delete a job.
- Display the status of the job queue processing system.

$SUBMIT starts (or restarts) job queue processing, if necessary, and then allows you to enter job queue commands.

For information on submitting jobs by a program, refer to *Event Driven Executive Language Programming Guide.*

Invoking $SUBMIT

You invoke $SUBMIT with the $L operator command or option 10.2 of the session manager.

You must start the job queue processor, $JOBQ, the first time you invoke $SUBMIT. You are prompted for a partition number in which to load $JOBQ. However, you do not need to select a partition. Just press the enter key and the operating system selects a partition large enough to contain $JOBQ. If you do not want to start the job queue processor, respond N and $SUBMIT ends.

```
> $L $SUBMIT

LOADING $SUBMIT     8P,09:18:62, LP=2300, PART=2

$SUBMIT - EDX JOB QUEUE SUBMISSION PROGRAM

ENTER '?' AT ANY TIME FOR HELP

THE JOB QUEUE PROCESSOR MUST BE LOADED TO CONTINUE.  LOAD IT (Y/N)?  Y

PARTITION NUMBER (1-8) OR ENTER FOR ANY:  4

THE JOB QUEUE PROCESSOR ($JOBQ) IS IN SUSPENDED MODE.
USE THE "RJQ" COMMAND IN $JOBQUT TO RESUME PROCESSING.

JOB QUEUE PROCESSOR LOADED INTO PARTITION 4

COMMAND (?):
```

$SUBMIT - Submit/Control Jobs in Job Queue Processor *(continued)*

$SUBMIT Commands

To display the $SUBMIT commands, enter a question mark in response to the prompting message COMMAND (?):

```
COMMAND (?):    ?

DJ  - DELETE A JOB
HJ  - HOLD A JOB
RJ  - RELEASE A JOB
SJ  - SUBMIT A JOB
SH  - SUBMIT A JOB AND HOLD IT
STQ - DISPLAY THE STATUS OF JOB QUEUE PROCESSING
DCH - DETACH $SUBMIT; RESUME-ATTN SUBMIT, END-ATTN ENDSUB

EN  - END $SUBMIT

COMMAND (?):
```

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

Each command and its explanation is presented in alphabetical order on the following pages.

DJ — Delete a Job

Use the DJ command to delete a job that is either waiting for execution or is being held. Identify the job to be deleted with the job number displayed when the job was submitted.

Note: To cancel an executing job, use the $C operator command.

Example: Delete job 5:

```
COMMAND (?):    DJ

JOB NUMBER:    5

ARE YOU SURE YOU WANT TO DELETE JOB 5?    Y
JOB 5 DELETED

COMMAND (?):
```

$SUBMIT - Submit/Control Jobs in Job Queue Processor *(continued)*

DCH — Detach $SUBMIT

Use the DCH command to place $SUBMIT in suspended mode. To resume processing, press the attention key, enter SUBMIT, and press the enter key. If you suspend $SUBMIT, the job queue processor remains active and any jobs executing complete processing.

To end $SUBMIT once it is in suspended mode, press the attention key and enter ENDSUB.

Example 1: Suspend $SUBMIT and then resume processing.

```
COMMAND (?):  DCH

>SUBMIT

COMMAND (?):
```

Example 2: End $SUBMIT.

```
COMMAND (?):  DCH

>ENDSUB

$SUBMIT ENDED
```

EN — End $SUBMIT

Use the EN command to end the $SUBMIT utility.

Example: End the $SUBMIT utility:

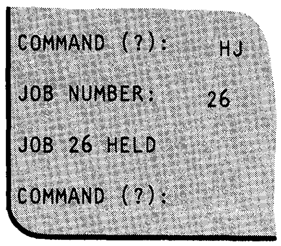
```
COMMAND (?):  EN

$SUBMIT ENDED
```

$SUBMIT - Submit/Control Jobs in Job Queue Processor *(continued)*

HJ — Hold a Job Waiting for Execution

Use the HJ command to place a job waiting for execution into a hold state. You must identify the job to be held with the job number displayed when the job was submitted. If you have submitted many jobs and cannot remember the job number of the specific job you wish to hold, use the STQ command to display a list of the jobs in the data set.

Example: Hold job 26 (a job already submitted) for later execution.



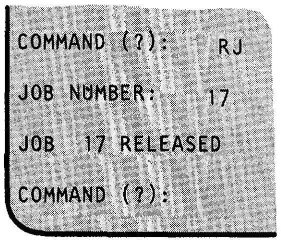
```
COMMAND (?):   HJ
JOB NUMBER:   26
JOB 26 HELD
COMMAND (?):
```

Once a job is placed in a hold state, you can release it for execution using the RJ command or delete it using the DJ command.

RJ — Release a Held Job

Use the RJ command to release for execution a job that is being held currently. To identify the job you wish to execute, enter the job number of the held job. Use the STQ command to display the status of individual jobs.

Example: Release job 17 for execution:



```
COMMAND (?):   RJ
JOB NUMBER:    17
JOB  17 RELEASED
COMMAND (?):
```

SJ — Submit Jobs to the Job Queue Processor

Use the SJ command to submit jobs for execution. You are prompted for the name of the $JOBUTIL command data set to be executed.

Optionally, you can assign a priority to the job you wish to execute. Four priorities are available, zero through three, where zero is the highest priority. Jobs with higher priorities execute before those with lower priorities. For example, if you want a particular job to run before other jobs, you can assign it a higher priority. On the other hand, you might not care how quickly a job executes. In this case, you do not assign a priority to the job. Just press the enter key, or enter an asterisk, in response to the JOB PRIORITY: prompt, and $SUBMIT assigns priority 2 to the job.

$SUBMIT - Submit/Control Jobs in Job Queue Processor *(continued)*

Example: Submit two jobs, one with and one without priority. The first job is submitted without a priority using prompt-reply mode; the second job is submitted specifying a priority of 1 using the single-line format. When using single-line format, enter the responses in the order the system expects them.

```
COMMAND (?):  SJ

ENTER COMMAND DATA SET (NAME,VOLUME):  COMPILE,CANPRC

ENTER A PRIORITY FROM 0 TO 3 (DEFAULT 2):

JOB PRIORITY DEFAULTED TO 2

JOB NUMBER    5 :  SUBMITTED
COMMAND DATA SET:  COMPILE,CANPRC
JOB PRIORITY    :  2

COMMAND(?):  SJ JOB1,EDX002 1

JOB NUMBER    6 :  SUBMITTED
COMMAND DATA SET:  JOB1,EDX002
JOB PRIORITY    :  1

COMMAND (?):
```

SH — Submit Job and Hold

Use the SH command to submit a job and place it into an immediate hold state. For example, a job may require specific diskette or tape data sets that are not available at the time you submit the job, or you may want to make sure a certain execution sequence is followed. By holding the job, you can set up the environment and then release the job for execution using the RJ command. If you decide to delete a held job, use the DJ command.

Example: Submit and hold a job with a priority of 3.

```
COMMAND (?):  SH

ENTER COMMAND DATA SET (NAME,VOLUME):  ASMJOB1,EDX002

ENTER A PRIORITY FROM 0 TO 3 (DEFAULT 2):  3

JOB NUMBER    7 :  SUBMITTED AND HELD
COMMAND DATA SET:  ASMOBJ1,EDX002
JOB PRIORITY    :  3

COMMAND (?):
```

$SUBMIT - Submit/Control Jobs in Job Queue Processor *(continued)*

STQ — Display the Status of Job Queue Processing

Use the STQ command to display the status of job queue processing and the status of individual jobs. STQ displays the jobs in the order in which they were submitted. For each job in the queue, $SUBMIT displays the following:

- Job number
- Command data set name
- Job priority
- Individual job status — waiting, held, executing.

Example: Display status of job queue processing.

```
COMMAND (?):    STQ

JOB QUEUE PROCESSING IS IN EFFECT

LOGGING TERMINAL IS $SYSLOG

JOB NUMBER | COMMAND DATA SET | PRIORITY | STATUS

    2        OVERNITE,USRVOL       2        HELD
    3        CALCJOB ,PAYROL       1        WAITING
    4        CHECKS  ,PAYROL       0        EXECUTING
    5        CALCJOB ,EDX002       0        WAITING
    6        JOB1    ,EDX002       2        WAITING
    7        ASMJOB1 ,EDX002       3        HELD

NUMBER OF JOBS QUEUED IS  6

COMMAND (?):
```

$S1ASM - Series/1 Assembler

$S1ASM assembles source programs coded in Series/1 assembler language into object modules. The source program can include Series/1 assembler language macros such as the commands provided by the Event Driven Executive Macro Library.

If you have a mixture of Series/1 assembler language statements and EDL instructions, you must use $S1ASM.

If your program contains only EDL instructions, use $EDXASM; your processing will be much faster. For more information, see the "$EDXASM - Event Driven Language Compiler" on page UT-265.

Before using $S1ASM, you must enter the source program onto a disk or diskette data set with one of the text editor utilities ($EDIT1N or $FSEDIT).

Required Data Sets

$S1ASM requires the following data sets. You must specify these data sets in the order shown when you use the $L operator command to invoke $S1ASM. The source program input data set and the object data set must be specified when you invoke $S1ASM using the session manager.

1. The *source program input data set (DS1)* contains the program to be assembled. Use a text editor ($EDIT1N or $FSEDIT) to create the source data set. You must specify the name of this data set when you invoke $S1ASM.

2. *Three work data sets (DS2-DS4)* are used by the assembler. They contain object code, relocation pointers, the symbol table, and other information. You must allocate these data sets if you use the $L operator command to invoke $S1ASM. For most programs, sizes of 3000, 3000, and 800 records for ASMWRK1, ASMWRK2, and ASMWRK3,, respectively, are sufficient. The sizes can increase when assembling a large program containing many macro calls. Specify the names of these data sets you allocated as a parameter on the $L operator command. These data sets are allocated automatically by the session manager.

3. The *object data set (DS5)* will contain the output object module from the assembly. It serves as input to the linkage editor. A size of 50 to 100 records is sufficient for most programs. You must allocate this data set and specify its name when you invoke $S1ASM.

Invoking $S1ASM

You invoke $S1ASM with the $L operator command, with option 2.3 of the session manager, or with the $JOBUTIL utility. In each case, you must provide the same information when you load $S1ASM for execution.

You can cancel the assembly or subsequent listing at any time by pressing the attention key and entering CA.

$S1ASM - Series/1 Assembler *(continued)*

Assembler Options

The $S1ASM assembler has several options. You specify the option(s) you want when you invoke $S1ASM. To direct the processing done by the assembler, the following assembler options are available.

LIST/NOLIST	LIST tells the assembler to write all assembly listings to the print file. NOLIST tells the assembler to write only the statistics and diagnostic messages, if any, to the print file.
TEXT/NOTEXT	TEXT tells the assembler to write the source and object program listings to the print file. NOTEXT suppresses this process.
ESD/NOESD	ESD tells the assembler to write the external symbol dictionary before the source program listing. NOESD suppresses this process.
RLD/NORLD	RLD tells the assembler to write the relocation dictionary after the source program listing. NORLD suppresses this process.
XREF/NOXREF/FULLXREF	XREF tells the assembler to write the cross-reference listing to the print file for only referenced symbols. NOXREF suppresses this process. FULLXREF tells the assembler to write the cross-reference listing to the print file for all defined and referenced symbols.
OBJECT/NOOBJECT	OBJECT causes the object module to be written to the OBJOUT data set. NOOBJECT suppresses this process.
MACRO/NOMACRO	MACRO tells the assembler to process any macros encountered in the source. NOMACRO suppresses this process.

$S1ASM - Series/1 Assembler *(continued)*

SYSPARM('...')	Defines up to 8 characters of information substituted for the &SYSPARM value during macro processing. Blanks may be contained within the apostrophes. Each blank counts as one character. A single apostrophe within the string must be represented by two apostrophes. The default value for &SYSPARM is a null character string.
LINECOUNT(n)	LINECOUNT specifies n as the number of lines per page of the print data set. If this option is omitted, the default line count is 55. The value of n must be in the range 1-999.
SETC(n)	SETC specifies n as the maximum number of characters allowed to be assigned to SETC symbols. The default value is assigned as 64 characters. The value must be in the range 2-98. If the assigned value is odd, it is rounded up to the next even number.
END	END terminates the option processing.
CA	CA cancels the assembly.

You must enter option(s) separated by commas. The default options are LIST, OBJECT, AND MACRO

Data Sets Used in Examples

For illustrative purposes, assume that:

- EDX002 is the IPL volume;
- Data sets ASMSRC and ASMOBJ are on the IPL volume;
- WORK contains the WK1, WK2, and WK3 data sets;
- PRNTR1 is the name of a print device; and
- $S1ASM is to have 32K bytes of dynamic storage allocated to it.

  Note: Unless you override the default dynamic storage size, $S1ASM uses 22K bytes of the partition for a work area. If space in the partition is severely limited, $S1ASM will execute with less dynamic storage work space. If more space is available, you can improve $S1ASM performance by specifying more dynamic storage work space. Dynamic work space for $S1ASM must be in the range of 8704 to 57344 bytes.

Assembling a Program Using the $L Operator Command

Assembling a program using the $L operator command involves loading the assembler, specifying macro libraries (if any), and specifying assembly options.

$S1ASM - Series/1 Assembler *(continued)*

Loading $S1ASM

When you assemble a program with the $L operator command, load $S1ASM from the ASMLIB volume.

The following example shows the prompt/reply sequence for invoking $S1ASM, specifying 32K of dynamic storage work area.

```
 > $L  $S1ASM,ASMLIB,32768
SOURCE  (NAME,VOLUME):    ASMSRC
WORK1   (NAME,VOLUME):    WK1,WORK
WORK2   (NAME,VOLUME):    WK2,WORK
WORK3   (NAME,VOLUME):    WK3,WORK
OBJECT  (NAME,VOLUME):    ASMOBJ

MACLIB1 (?):
```

The following single-line entry is equivalent to the multiline entries above.

```
 > $L $S1ASM,ASMLIB,32768 ASMSRC WK1,WORK WK2,WORK WK3,WORK ASMOBJ
MACLIB1 (?):
```

Specifying Macro Libraries

If you require macros for an assembly, you can specify one or two macro library volumes.

In both of the previously shown methods of loading $S1ASM, you are then prompted as follows:

```
MACLIB1 (?):
```

A null response (ENTER) causes the next prompt (ENTER OPTIONS) to appear. If you specify a macro library for the MACLIB1 prompt (as shown in the following example), another prompt (MACLIB2) appears:

```
MACLIB1 (?):   MACLB1
MACLIB2 (?):
```

You can supply the name of another macro library or enter a null response.

Note: If your source program contains Event Driven Language instructions, you must specify, as either MACLIB1 or MACLIB2, the name of the volume(s) that contains the Event Driven Executive Macro Library and copy code.

$S1ASM - Series/1 Assembler *(continued)*

Specifying Assembly Options

After entering the macro library volume name(s) or a null response, you are prompted as follows:

```
ENTER OPTIONS (?):
```

After the prompt appears, you can:

- Enter a null response and take the default options (LIST, OBJECT, MACRO).
- Enter the options you desire (for example, LIST, NOXREF, NORLD). See "Assembler Options" on page UT-510.
- Enter a question mark to display a list of the options. After the options are displayed, the ENTER OPTIONS (?): prompt is displayed. Enter the option(s) you want.
- Enter the options, followed by the name of your output device.

If you do not specify the name of the output device with your options, as follows:

```
ENTER OPTIONS (?):  LIST,NOXREF,NORLD
```

you are prompted with:

```
ENTER OUTPUT DEVICE NAME:
```

You can enter the name of the device on which your listings and diagnostic messages are to be written. You can specify an asterisk (*) to direct the listings back to your loading terminal. If you do not specify an output device (a null response), the output device defaults to $SYSPRTR.

The next message $S1ASM displays is:

```
CPA000I ASSEMBLER STARTED
```

$S1ASM - Series/1 Assembler *(continued)*

Example: The following examples shows how to use single-line entry and prompt-mode entry and use:

- ASMSRC for the source library
- WK1, WK2, and WK3 for work files
- ASMOBJ for the output file
- MACLB1 for the macro library
- PRNTR1 for the output listing.

Single-line Format: This example illustrates how to specify multiple data sets/volumes using single-line entry.

```
 > $L $S1ASM,ASMLIB ASMSRC WK1,WORK WK2,WORK WK3,WORK ASMOBJ
MACLIB1 (?):    MACLB1
MACLIB2 (?):
ENTER OPTIONS (?):    LIST,NOXREF,NORLD PRNTR1
CPA000I ASSEMBLER STARTED
```

Prompt/Reply Format: This example illustrates how to specify multiple data sets/volumes in prompt mode.

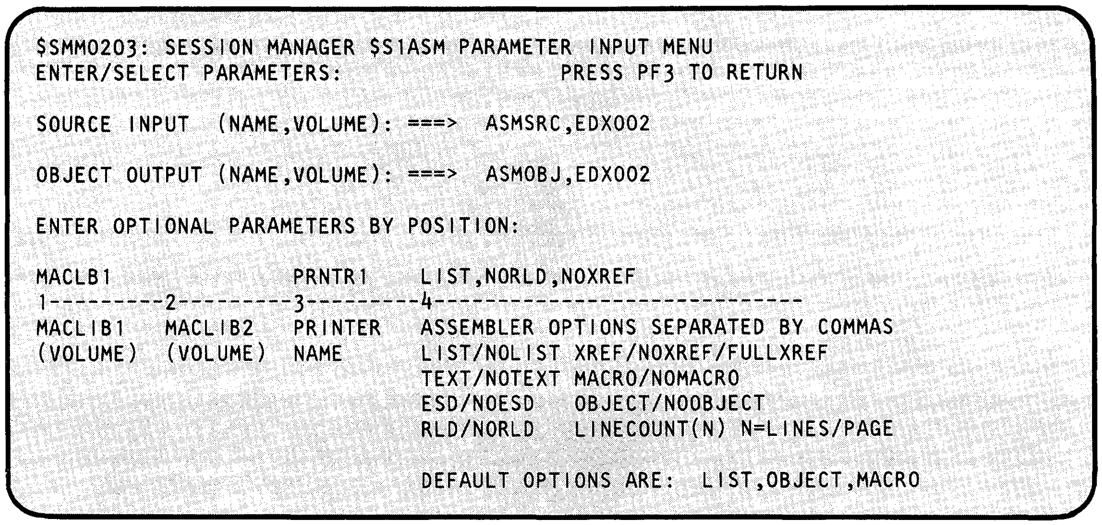
```
 > $L $S1ASM,ASMLIB
SOURCE (NAME,VOLUME):    ASMSRC
WORK1  (NAME,VOLUME):    WK1,WORK
WORK2  (NAME,VOLUME):    WK2,WORK
WORK3  (NAME,VOLUME):    WK3,WORK
OBJECT (NAME,VOLUME):    ASMOBJ
MACLIB1 (?):    MACLB1
MACLIB2 (?):
ENTER OPTIONS (?):    LIST,NOXREF,NORLD
ENTER OUTPUT DEVICE NAME:    PRNTR1
CPA000I ASSEMBLER STARTED
```

$S1ASM - Series/1 Assembler *(continued)*

Assembling a Program Using the Session Manager

To invoke $S1ASM using the session manager, select option 2 from the program preparation secondary option menu.

Figure 30 shows the input menu for entry of the required data sets and optional parameters. Source data set ASMSRC is to be assembled and the object output is directed to data set ASMOBJ.



```
$SMM0203: SESSION MANAGER $S1ASM PARAMETER INPUT MENU
ENTER/SELECT PARAMETERS:                 PRESS PF3 TO RETURN

SOURCE INPUT  (NAME,VOLUME): ===>  ASMSRC,EDX002

OBJECT OUTPUT (NAME,VOLUME): ===>  ASMOBJ,EDX002

ENTER OPTIONAL PARAMETERS BY POSITION:

MACLB1              PRNTR1    LIST,NORLD,NOXREF
1---------2---------3---------4----------------------------
MACLIB1   MACLIB2   PRINTER   ASSEMBLER OPTIONS SEPARATED BY COMMAS
(VOLUME)  (VOLUME)  NAME      LIST/NOLIST XREF/NOXREF/FULLXREF
                              TEXT/NOTEXT MACRO/NOMACRO
                              ESD/NOESD   OBJECT/NOOBJECT
                              RLD/NORLD   LINECOUNT(N) N=LINES/PAGE

                              DEFAULT OPTIONS ARE:  LIST,OBJECT,MACRO
```

Figure 30. $S1ASM parameter input menu

$S1ASM - Series/1 Assembler *(continued)*

Assembling a Program Using $JOBUTIL

You can invoke $S1ASM with the job stream processor $JOBUTIL. The same options are available through the PARM facility of $JOBUTIL as were described previously under the $L command. If you require the default options, you can leave the PARM card blank, but you must include it in the procedure.

The following is a sample procedure:

```
PROGRAM $S1ASM,ASMLIB
DS      ASMSRC
DS      WK1,WORK
DS      WK2,WORK
DS      WK3,WORK
DS      ASMOBJ
PARM     MACLB1              PRNTR1    LIST,NOXREF,NORLD
EXEC
```

Note: Parameters of the PARM statement are column- dependent:

MACLB1,if coded, must start in column 10.
MACLB2, if coded, must start in column 20.
PRNTR1, if coded, must start in column 30.
The option list, if coded, must start in column 40.

$S1ASM Output

Upon successful completion of $S1ASM, the object data set will contain the output of the assembler. For a description of that output, refer to the *IBM Series/1 Event Driven Executive Macro Assembler Reference.*

$S1S1UT1 - Series/1-to-Series/1

You can use $S1S1UT1 to invoke the IPL feature and perform data transfer and status operations.

After the Series/1-to-Series/1 Attachment is installed, use the EC (echo test) command of $S1S1UT1 to verify that the attachment is installed correctly.

$S1S1UT1 can also be used to check out an application. For example, if the application on the initiating processor issues a PRINTEXT, the RE (read) command of $S1S1UT1 can be used on the responding processor to read the data from the PRINTEXT.

Invoking $S1S1UT1

You invoke $S1S1UT1 with the $L operator command or option 4.8 of the session manager. $S1S1UT1 must be active on two connected processors for processor-to-processor communication via the RE (read), WR (write), and AB (abort) commands. For example, if you issue a WR command on one processor, then a corresponding RE command must be issued on the other processor.

When you load $S1S1UT1, it immediately issues a DD (define device) command to obtain the terminal name.

```
> $L $S1S1UT1
LOADING $S1S1UT1   38P,02:52:52, LP= 0000, PART= 2
"?" FOR LIST OF COMMANDS

ENTER S1S1 DEVICE NAME:
```

$S1S1UT1 Commands

To display the $S1S1UT1 commands at your terminal, enter a question mark in response to the prompting message COMMAND (?):.

```
COMMAND(?):  ?

AB  PERFORM WRITE ABORT
DD  DEFINE DEVICE NAME
IP  IPL OTHER PROCESSOR
RE  READ DATA
RS  RESET DEVICE
WR  WRITE DATA
EN  END THE PROGRAM
EC  ECHO TEST
ST  OBTAIN STATUS

COMMAND(?):
```

After the $S1S1UT1 displays commands, you are again prompted with COMMAND(?):. Then you can respond with the command you want (for example, AB). Each command and its explanation is presented in alphabetical order on the following pages.

$S1S1UT1 - Series/1-to-Series/1 *(continued)*

AB — Perform Write Abort

Use the AB command to issue a "write abort" for a pending operation on the initiating processor. $S1S1UT1 issues the write abort operation to the attachment specified by the most recent DD command. This command is used by the responding processor to end a data transfer operation abnormally.

Example:

```
COMMAND(?): AB
ABORT OPERATION? Y

COMMAND(?):
```

DD — Define Device Name

Use the DD command to specify the terminal name of the attachment used for all ENQTS of the attachment for subsequent operations, until you issue another DD command. The terminal name is the name specified on the TERMINAL configuration statement at system generation.

Example:

```
COMMAND(?): DD
S1S1 DEVICE NAME: S1S100

COMMAND(?):
```

EC — Echo Test

Use the EC command to verify that the attachment is installed correctly. It results in a continuous exchange of 1024-byte records between the attached processors. To terminate EC press the attention key and enter "EC."

Example:

```
COMMAND(?): EC
ECHO EXERCISE
ATTN "EC" TO STOP ECHO TEST
ATTEMPTING TO SYNCH UP
1K DATA TRANSMITTED
1K DATA RECEIVED
DATA CHECKS OUT!

(Attention)
> EC

COMMAND(?):
```

$S1S1UT1 - Series/1-to-Series/1 *(continued)*

EN — End the Program

Use the EN command to end the $S1S1UT1 utility.

Example:

```
COMMAND(?):  EN

$S1S1UT1 ENDED AT 00:00:00
```

IP — IPL the Other Processor

Use the IP command to issue an IPL (initial program load) command to the secondary processor, if it is the primary processor issuing the IP command. $S1S1UT1 prompts you for the member name and volume that contain the nucleus to be transferred to the secondary processor.

The nucleus being transferred must begin with the characters $EDXNUC. It must be a single partition supervisor with no overlays. The IPL bootstrap program, IPLS1S1, must be located in the IPL volume; $S1S1UT1 prompts you to verify that the volume name specified is correct. You are also asked if the secondary system has a disk or diskette device and for the address of that device. The specified disk or diskette becomes the default direct-access device for disk I/O operations on the processor being initialized. The bootstrap program is sent to the slave and it reads the specified nucleus, 1024 bytes at a time, across the attachment. Control is passed to the nucleus upon completion of the transfer.

Example:

```
COMMAND(?):  IP

ENTER NUCLEUS DSN:  $EDXNUC

ENTER NUCLEUS VOLUME:  EDX002
IS THERE A DEFAULT DISK DEVICE FOR THE NUCLEUS?  Y
SUPPLY DISK/DISKETTE DEVICE ADDRESS(HEX):  48
IPL PROGRAM NAME:  IPLS1S1  ON VOLUME:  EDX002
OK?  Y
READY FOR IPL?  Y
```

$S1S1UT1 - Series/1-to-Series/1 *(continued)*

RE — Read Data

Use the RE command to issue an ENQT instruction for the terminal name specified by the most recent DD command. $S1S1UT1 then issues a READTEXT instruction for that terminal to read data from the other processor. If $S1S1UT1 is active on the other processor, a WR command must be issued to complete an RE command.

Example:

```
COMMAND(?):  RE

MESSAGE RECEIVED!

THIS IS TEST DATA RECEIVED

COMMAND(?):
```

RS — Reset Device

Use the RS command to issue a device reset to the attachment specified by the most recent DD command. This command clears any pending interrupt or busy condition.

Example:

```
COMMAND(?):  RS
RESET ATTACHMENT?  Y

COMMAND(?):
```

ST — Obtain Status

Use the ST command to obtain status information on an operation. The status returned by this command is the same as the information returned when a TERMCTRL STATUS instruction is issued.

Example:

```
COMMAND(?):  ST
OBTAIN CYCLE STEAL STATUS ALSO?  Y
WAIT FOR HEADER?  N
HEADER WORDS:  1100 0400
READ(READTEXT) ISSUED BY OTHER CPU
NO. BYTES = 0400
DIAGNOSTIC JUMPER WORD:  02E6
CYCLE STEAL STATUS:        (11 words of status)

COMMAND(?):
```

$S1S1UT1 - Series/1-to-Series/1 *(continued)*

WR — Write Data

Use the WR command to issue an ENQT instruction to the terminal name specified by the most recent DD command. A PRINTEXT instruction is then issued for that terminal to write data to the other processor. If $S1S1UT1 is active on the other processor, an RE command must be issued to complete a WR command.

Example:

```
COMMAND(?):  WR
ENTER TEXT:
TEST DATA TO WRITE

MESSAGE SENT!

COMMAND(?):
```

$TAPEUT1 - Tape Management

$TAPEUT1 performs several commonly-used tape management functions. You can initialize tapes, allocate tape data sets, copy data sets or volumes to or from tape, copy tape to tape, print tape records, dump/restore disk devices, and test the tape transport hardware.

Invoking $TAPEUT1

You invoke $TAPEUT1 with the $L operator command or option 3.10 of the session manager. Once invoked and prior to accepting commands, $TAPEUT1 displays the following information about the tapes defined to the system:

- tape identification
- density selection for the 4969 (800, 1600, DUAL) or the 4968 (1600, 3200, DUAL)

  Note: The 3200 density for the 4968 is not ANSI compatible but is compatible with other 4968 tape units.

- label type (SL: standard label, NL: null label, BLP: bypass label processing)
- current density setting
- tape speed (inches per second) - 4968 only
- online or offline
- volume information (if an online SL tape)
- device address.

Example: Loading $TAPEUT1 and its automatic display for system tapes.

```
> $L $TAPEUT1
LOADING $TAPEUT1    21P,11:11:33, LP= 0000, PART=2
TAPE01 DUAL SL 1600 OFFLINE
   DEVICE ADDRESS = 004C
TAPE02 DUAL NL 1600 OFFLINE
   DEVICE ADDRESS = 004D

COMMAND (?):
```

Once you have invoked $TAPEUT1, you can list this information at any time by using the LT (list tape drives and attributes) command.

Note: Error logging of tape errors is available. Refer to the *Problem Determination Guide.*

$TAPEUT1 - Tape Management *(continued)*

$TAPEUT1 Commands

To display the $TAPEUT1 commands at your terminal, enter a question mark in reply to the prompting message COMMAND (?):.

```
COMMAND (?):  ?

CD ---- COPY TAPE DATA SET
CT ---- CHANGE TAPE ATTRIBUTES
DP ---- DUMP TAPE
EN ---- END $TAPEUT1
EX ---- EXERCISE TAPE
IT ---- INITIALIZE TAPE
LT ---- LIST TAPE DRIVES AND ATTRIBUTES
MT ---- MOVE TAPE
RC ---- DISPLAY TAPE RETURN CODES
RT ---- RESTORE DISK/VOL FROM TAPE
ST ---- SAVE DISK/VOL ON TAPE
TA ---- ALLOCATE TAPE DATA SET

COMMAND (?):
```

After $TAPEUT1 displays the commands, it prompts you with COMMAND (?): again. Then you can respond with the command of your choice (for example, CD). Each command and its explanation is presented in alphabetical order on the following pages.

CD — Copy Data Set

Use the CD command to:

- copy a disk or diskette data set onto a tape
- copy a tape data set into a disk or diskette data set
- copy a tape data set onto another tape.

CD writes a trailer label at the end of the data set on the target tape if it is a standard label tape. The system does not write header labels on standard or nonlabeled tapes; therefore, you must preallocate the target tape data set.

If you are copying a disk or diskette data set to tape, the tape records are 256 bytes. If you copy a tape data set from another system (for example, an S/370) to a disk or diskette and the source records are not 256 bytes, the system splits the source records into multiple 256-byte records and pads any unused bytes with zeros. Prior to copying, $TAPEUT1 prompts you for the maximum input record size. If the actual record size differs from the input record size, $TAPEUT1 prompts if you wish to continue processing. $TAPEUT1 issues this prompt every time it encounters a record with the wrong length.

$TAPEUT1 - Tape Management *(continued)*

Consider the following when you are copying data sets:

- When you reach a tapemark (end of input data), $TAPEUT1 prompts you to continue. If you have more records to copy, you can continue; however, make sure that the target tape has sufficient room. $TAPEUT1 prompts at every tapemark it encounters on the source tape. If you do not wish to continue, the system writes the trailer label on the target tape.
- To copy the contents of one tape to another tape, thereby creating an exact duplicate of the entire tape (header label and data records or only data records), you can use either of two methods:
  - To copy only data records, initialize the target tape (using the IT command) so that it has the same label type as the source tape. Copy (using the CD command) the source tape to the target tape. This allows you to create a new header label on the target tape and to duplicate only the data records from the source tape.
  - To create an exact duplicate of the source tape, mount the source and target tapes on drives specified for bypass label processing. Then copy (using the CD command) the entire source tape. The target tape becomes an exact duplicate of the source (all label records, all data records, and all trailer labels).
- If the source data set is larger than the target tape, you can continue copying onto another tape.
- If the source data set is larger than the target disk data set, you can continue copying into another disk data set.
- You can also combine a multiple disk data set into one tape data set.

$TAPEUT1 - Tape Management *(continued)*

Example 1: Copy data from a disk to a tape (standard label tape).

```
COMMAND (?):  CD

SOURCE (NAME,VOLUME):  $TAPEUT1,EDX002
TARGET (NAME,VOLUME):  DATA1111,123456
ENTER SOURCE BLOCKSIZE (NULL=DISK(ETTE)):
USE ATTN/CA TO CANCEL COPY

ARE ALL PARMS CORRECT? (Y,N):  Y
EOD ON SOURCE DATA SET
    25 RECORDS COPIED

COMMAND (?):
```

Example 2: Copy data from a tape (nonlabeled) to a tape (standard labeled).

```
COMMAND (?):  CD

SOURCE (NAME,VOLUME):  X,TAPE01
TARGET (NAME,VOLUME):  DATA1111,123456
ENTER SOURCE BLOCKSIZE (NULL=DISK(ETTE)):
USE ATTN/CA TO CANCEL COPY

ARE ALL PARMS CORRECT? (Y,N):  Y
EOD ON SOURCE DATA SET
    25 RECORDS COPIED

COMMAND (?):
```

Note: TAPE01 is the ID assigned to the tape drive at system generation.

$TAPEUT1 - Tape Management *(continued)*

CT — Change Tape Drive Attributes

Use the CT command to reset the label type and density for any tape drive. The label type and density are set at system generation. CT allows you to reconfigure the tape drives dynamically. You must vary the tape drive offline before you can change its attributes.

```
CT id
```

where id is the tape ID assigned to the tape drive during system generation.

For the 4969, the system displays the current settings for label and density. For the 4968, the system displays the current settings for label, density, and tape speed. $TAPEUT1 prompts you to enter any changes.

Note: For the 4968 only, a density selection of 1600 causes the system to set the tape drive speed to 25 inches per second (IPS). A density selection of 3200 results in a tape drive speed of 50 inches per second (IPS).

The CT command fails and issues an error message on the terminal for the following conditions:

- invalid device address
- tape drive is not varied offline
- invalid label type
- invalid density
- tape drive not defined as dual density if you tried to change the density.

Example 1: Specify changes to the label processing and density selection for a 4969 tape drive at address 4C.

```
COMMAND (?):  CT

ENTER TAPEID (1-6 CHARS):  TAPE01
TAPE TAPE01 AT ADDR 4C IS BLP 1600 BPI
DO YOU WISH TO MODIFY (Y OR N)?:  Y
LABEL (NULL,SL,NL,BLP)?:  SL
DENSITY (NULL,800,1600):  800
TAPE TAPE01 AT ADDR 4C IS SL 800 BPI

COMMAND (?):
```

$TAPEUT1 - Tape Management *(continued)*

Example 2: The system makes no changes for label processing and density selection for a 4969 tape drive at address 4D.

```
COMMAND (?):  CT TAPE02

TAPE TAPE02 AT ADDR 4D IS SL 800 BPI
DO YOU WISH TO MODIFY (Y OR N)?:  N

COMMAND (?):
```

DP — Dump Tape Records

Use the DP command to dump tape record(s) on the system printer ($SYSPRTR), on the terminal from which you invoked $TAPEUT1, or on the terminal you specify. The output format is similar to the $DISKUT2 utility: hexadecimal data plus the EBCDIC conversion.

The system shows the record number for each record and prints the words 'tapemark' when it encounters a tapemark. $TAPEUT1 prompts you for the number of records you want to print. When the system detects a tapemark, $TAPEUT1 prompts you to continue or to end the dump.

$TAPEUT1 prompts you for the maximum record size. This allows you to print record sizes up to the maximum amount of storage available. If the record you are reading is smaller or larger than the maximum size specified, the system issues a message indicating:

- wrong-length record
- the actual record size

The system prints smaller records and pads them with zeros for their length. It truncates larger records and prints to the maximum size allowed. $TAPEUT1 informs you of the actual record size.

Notes:

1. The tape must be varied offline before you can dump records.
2. The TAPEID you specify must match the one specified at system generation.
3. This is an offline utility; therefore, the tape will not rewind automatically when the dump finishes. Use the move tape command (MT) to rewind the tape.
4. You can use the dump command (DP) and the move tape command together (MT) to search and dump portions of a tape selectively. DP keeps track of the actual record count (along with the MT command) and will print this value. However, if you issue a forward space file (FSF) control function to move the tape to the end of the file and then issue a back space records (BSR) control function to back up the tape a specified number of records, DP cannot track the actual record count. As a result, it prints a relative record count with an R printed after the number indicating the count is relative.

$TAPEUT1 - Tape Management *(continued)*

Example: Dump five records on display terminal.

```
COMMAND (?):  DP

ENTER MAXIMUM BLOCKSIZE:  256
USE ATTN/CA TO CANCEL DUMP
ENTER TAPEID (1-6 CHARS):  TAPE01
ENTER NUMBER OF RECORDS TO DUMP:  5
PRINT DUMP ON $SYSPRTR? (Y,N):  N
ENTER TARGET TERMINAL NAME (*,NAME):  *

    (The system directs the dump to the terminal
    where you are currently assigned)

COMMAND (?):
```

If you respond Y to the PRINT DUMP ON $SYSPRTR? prompt, the system directs the dump to $SYSPRTR If you respond N, $TAPEUT1 prompts you for the terminal name of your choice. You can respond as follows:

- * = terminal where you are currently assigned
- name = device name of the target terminal.

If the tape you want to dump is in use, $TAPEUT1 prompts you as follows:

```
DEVICE NOT OFFLINE
DO YOU WISH TO CONTINUE? (Y,N):
```

If you specify Y, the dump will continue but the device will be unusable by the original user.

EX — Exercise Tape

EX, a software exerciser, performs two operations:

- It exercises any of the three label-type tapes to ensure that the I/O commands to that tape are executing correctly.
- It analyzes the surface of the tape to verify that the surface is free of defects. The system prints on the system printer ($SYSPRTR) any errors and their approximate location on the tape.

Note: Surface analysis writes records over the information currently on the tape. This destroys any existing data records or labels.

$TAPEUT1 - Tape Management *(continued)*

Each operation is optional and the system prompts you before it continues.

The EX command performs the following functions:

- Writes 600 unique records to a data set on the tape
- Closes and reopens the data set
- Finds a particular record within the data using NOTE/POINT
- Verifies that it accesses the correct record
- Performs a surface analysis of the tape by writing over the tape.

If it encounters an error, the system prints the return code and the contents of the buffer. The buffer contains all FFFF's except for the last word which is the record count of the failing record.

Example: Exercise tape MYDATA.

```
COMMAND (?):  EX
TARGET (NAME,VOLUME):  MYDATA,123456
USE ATTN/CA TO CANCEL THE EXERCISER
DO YOU WANT TO EXERCISE THE SOFTWARE (Y/N):  Y
   ...
  Exerciser runs and prints status on printer
   ...
WRITE ENTIRE SURFACE OF TAPE? (Y/N):  Y

THIS SECTION WILL DESTROY ALL LABEL FIELDS
DO YOU WISH TO CONTINUE (Y,N):  Y
   ...
  Exerciser writes on entire surface of tape
   ...
COMMAND (?):
```

A sample of the data the EX command prints follows.

$TAPEUT1 - Tape Management *(continued)*

```
TAPE EXERCISER STARTED
WRITE 600 UNIQUE RECORDS TO THE TAPE
TEST DATA WRITTEN
STEP        1 SUCCESSFUL
CLOSE DATA SET FOR REUSE VIA SETEOD
STEP        2 SUCCESSFUL
READ 250 OF THE RECORDS
NOTE PRESENT POSITION
STEP        3 SUCCESSFUL
POINT TO RECORD 150 AND READ THAT RECORD
STEP        4 SUCCESSFUL
NOTE PRESENT POSITION
STEP        5 SUCCESSFUL
VERIFY DATA RECORD
STEP        6 SUCCESSFUL
POINT TO RECORD 500 AND READ THAT RECORD
STEP        7 SUCCESSFUL
 .....
 .....
 .....
VERIFY 2nd WORD OF TCB CONTAINS ACTUAL BLOCKSIZE
STEP       17 SUCCESSFUL
READ UNTIL END OF DATA ENCOUNTERED
VERIFY LAST DATA RECORD
STEP       18 SUCCESSFUL
CLOSE DATA SET FOR REUSE
STEP       19 SUCCESSFUL
REOPEN THE DATA SET
READ 598 OF THE RECORDS
ATTEMPT TO READ MORE RECORDS WITH 1 STMT THAN ARE AVAILABLE
VERIFY END-OF-DATA RETURN CODE
STEP       20 SUCCESSFUL
VERIFY 2ND WORD OF TCB CONTAINS ACTUAL # REALLY READ
STEP       21 SUCCESSFUL
WRITE ENTIRE SURFACE OF TAPE?
REWIND TAPE TO LOAD POINT
STEP       22 SUCCESSFUL
WRITE ENTIRE TAPE
WRITE TAPE MARKS ON END OF TAPE
STEP       23 SUCCESSFUL
REWIND TAPE TO LOAD POINT
STEP       24 SUCCESSFUL
CLOSE TAPE OFFLINE
STEP       25 SUCCESSFUL
```

Figure 31. Sample of output from EX (tape exerciser).

$TAPEUT1 - Tape Management *(continued)*

IT — Initialize Tape

Use the IT command to initialize a new tape completely or to change the label information on a used tape. The IT command initializes tapes for nonlabeled and standard label use.

When you initialize a tape as nonlabeled, the IT command writes three tapemarks, deleting any previous labels on the tape.

When you initialize a tape as standard label, the IT command writes on the tape:

- a volume label (VOL1)
- a header label (HDR1)
- 2 tapemarks to delimit the label information and to indicate an empty data set
- a trailer label (EOF1) and 2 tapemarks signifying the end of data on the tape.

$TAPEUT1 prompts you for the the contents of all required label fields.

Example: Initialize a standard label tape.

```
COMMAND (?):  IT

ENTER TAPEID (1-6 CHARS):  TAPE01
STANDARD LABEL 1600 BPI?  Y
TAPEDS (NAME,VOLUME):  DATA1111,123456
OWNERID (1-10 CHARS):  OWNER-ID
EXPIRATION DATE (YYDDD):  79001
TAPE INITIALIZED

COMMAND (?):
```

Your tape must be varied offline before you can initialize it. If the tape is not offline, $TAPEUT1 issues the following warning message:

```
WARNING! DEVICE ISN'T OFFLINE.
CONTINUE?
```

If you respond Y, $TAPEUT1 varies the tape offline, initializes it, and varies it online under its new name. If you respond N, the IT command ends and $TAPEUT1 prompts you for another command.

$TAPEUT1 - Tape Management *(continued)*

If you are changing the label information on a used tape that has an unexpired data set, $TAPEUT1 prompts you as follows:

```
UNEXPIRED DATA SET!
CONTINUE?
```

If you respond Y, $TAPEUT1 varies the tape offline, initializes it, and varies it online under its new name. If you respond N, the IT command ends and $TAPEUT1 prompts you for another command.

Notes:

1. The system initializes your tape with the same attributes (label and density) as those defined for the tape drive on which you mounted the tape. Refer to the TAPE statement in the *Installation and System Generation Guide.*

2. If you initialize a 4968 tape drive with a density of 3200 BPI, the initialized tape will not be ANSI compatible but is compatible with any other 4968 tape unit.

3. When specifying the volume and data set names, do not use the same names as were specified for tape ID at system generation.

You can also initialize a tape from a program or by invoking $JOBUTIL and passing the following parameters:

Name of module: $TAPEIT

Required parameters:

	LENGTH	NL	SL	EXPLANATION
$PARM1	3 words	X	X	TAPEID
$PARM4	4 words	X		DS name
$PARM8	3 words	X		Volume
$PARM11	5 words	X		Owner ID
$PARM16	3 words	X		Create date (YYDDD) (if no timers in system)
$PARM19	3 words		X	Expiration date(YYDDD)

Examples of loading $TAPEIT from a program or through a $JOBUTIL procedure follow.

$TAPEUT1 - Tape Management *(continued)*

Example 1: Loading $TAPEIT from a program for a standard label tape; timers are included in the system.

```
   .
   .
   .
LOAD $TAPEIT,PARMS,EVENT=ITWAIT
WAIT ITWAIT
   .
   .
   .
PARMS EQU *    $TAPEIT PARAMETER LIST
ID      DATA      CL6'TAPE01'     (TAPEID - 3 WORDS)
DSN     DATA      CL8'MYDATA'     (DATA SET NAME - 4 WORDS)
VOL     DATA      CL6'10010'      (VOLUME NAME - 3 WORDS)
OWNR    DATA      CL10'SMITH'     (OWNER ID - 5 WORDS)
CRDATE  DATA      CL6' '          (UNUSED CREATE DATE -
*                                  3 WORDS)
EXDATE  DATA      CL6'81100 '     (EXPIRATION DATA -
*                                  3 WORDS)

* CREATION DATE AND EXPIRATION DATE ARE YYDDD FORMAT
* AND MUST BE LEFT JUSTIFIED WITHIN A 6-BYTE FIELD.
```

Example 2: The following figures show the procedures you can set up to initialize tapes through the $JOBUTIL utility.

Figure 32 is a procedure to initialize a standard label tape. In the PARM statement, you specify the TAPEID, data set name, volume, owner identification, create date, and expiration date.

```
REMARK
REMARK   DEFINE -TAPE01- FOR SL,1600 BPI
REMARK   MOUNT A TAPE ON -TAPE01- BUT DO *NOT* VARY ON
PAUSE    ***** IS THE TAPE MOUNTED? *****(REPLY -GO-)
PROGRAM  $TAPEIT
*******************************************************
PARM     TAPE01TEMPDATA123456DEPT563   76100 78100
EXEC
JUMP     OK1,EQ,-1
```

Figure 32. $JOBUTIL procedure for invoking $TAPEIT for an SL tape.

$TAPEUT1 - Tape Management *(continued)*

Figure 33 is a procedure to initialize a nonlabeled tape. When you initialize a tape as nonlabeled, the system writes three tapemarks on the tape deleting any previous labels on the tape.

```
REMARK
REMARK    **** DEFINE -TAPE01- FOR 800 BPI & NL PROCESSING
REMARK    MOUNT A TAPE ON -TAPE01- BUT DO *NOT* VARY IT ON
PAUSE     ***** IS THE TAPE MOUNTED? *****(REPLY - GO-)
PROGRAM   $TAPEIT
*****************************
PARM      TAPE01
EXEC
JUMP      OK4,EQ,-1
```

Figure 33. $JOBUTIL procedure for invoking $TAPEIT for an NL tape.

Figure 34 is a procedure to test the error code $TAPEIT returns.

```
REMARK    POST CODE 102 - DEVICE NOT OFFLINE
PROGRAM   $TAPEIT
PARM      TAPE01
EXEC
JUMP      ERR3,EQ,102
```

invoked through $JOBUTIL.

Figure 34. Testing the post code returned by $TAPEIT when

If you initialize a tape by loading $TAPEIT from a program or by invoking $JOBUTIL and passing the above parameters, the system returns following post codes.

Code	Condition
-1	Function successful.
RC	Any tape I/O return code.
101	TAPEID not found.
102	Device not offline.
103	Unexpired data set on tape.
104	Cannot initialize BLP tapes.

$TAPEUT1 - Tape Management *(continued)*

LT — List Tape Drives and Attributes

Use the LT command to list all the tape drives defined to the system and the attributes of each on the terminal where you invoked $TAPEUT1. The information displayed for each tape drive is as follows:

- TAPEID
- density selection for the 4969 (800, 1600, DUAL)
- density selection for the 4968 (1600, 3200, DUAL)
- label type (SL, NL, BLP)
- current density setting
- current tape speed (4968 only)
- status (online or offline)
- volume information (if an online SL tape)
- device address

Example:

```
COMMAND (?):    LT

TAPE DUAL SL 1600 OFFLINE
 DEVICE ADDRESS = 004C
TAPE DUAL NL 1600 OFFLINE
 ADDRESS = 004D

COMMAND (?):
```

$TAPEUT1 - Tape Management *(continued)*

MT — Move Tape

Use the MT command to control tape motion on the specified tape. The available control functions are:

BSF - Back space file

BSR - Back space records

FSF - Forward space file

FSR - Forward space records

OFF - Set device offline

REW - Rewind

ROFF - Rewind offline

WTM - Write tapemark

A count is available for FSR, BSR, FSF, BSF, and WTM so that you can specify the number of records or files you want to space over or the number of tapemarks you want to write.

An EOT (end-of-tape) terminates only the write tapemark (WTM) function and issues a return code. If you wish to proceed past the EOT, you must request another WTM.

Notes:

1. You can proceed past the EOT, but make sure that there is sufficient tape to perform the operation.

2. The tapemark record is smaller than the end-of-tape (EOT) foil indicator so you could receive two or more end-of-tape strip indications for the same EOT.

A tapemark ends FSR or BSR operations and the system positions the tape following that tapemark.

Example: Move TAPE01 forward 3 records.

```
COMMAND (?):  MT

ENTER TAPEID (1-6 CHARS):  TAPE01
TYPE? FSR/BSR/FSF/BSF/WTM/REW/ROFF/OFF:  FSR 3
* the action occurs *
FSR SUCCESSFUL
TYPE? FSR/BSR/FSF/BSF/WTM/REW/ROFF/OFF:  END

COMMAND (?):
```

$TAPEUT1 - Tape Management *(continued)*

If the tape is positioned at the first record and the utility forward spaces 3 records, it positions the tape at the 4th record.

Notes:

1. You must vary your tape offline before you can issue the motion commands. If you have not varied the tape offline, the system issues a warning message and $TAPEUT1 prompts you to continue.

2. The response to ENTER TAPEID must be the same TAPEID that you specified at system generation.

3. This is an offline utility; therefore, the system will not reposition or rewind the tape when it ends. Use the REW command to rewind the tape.

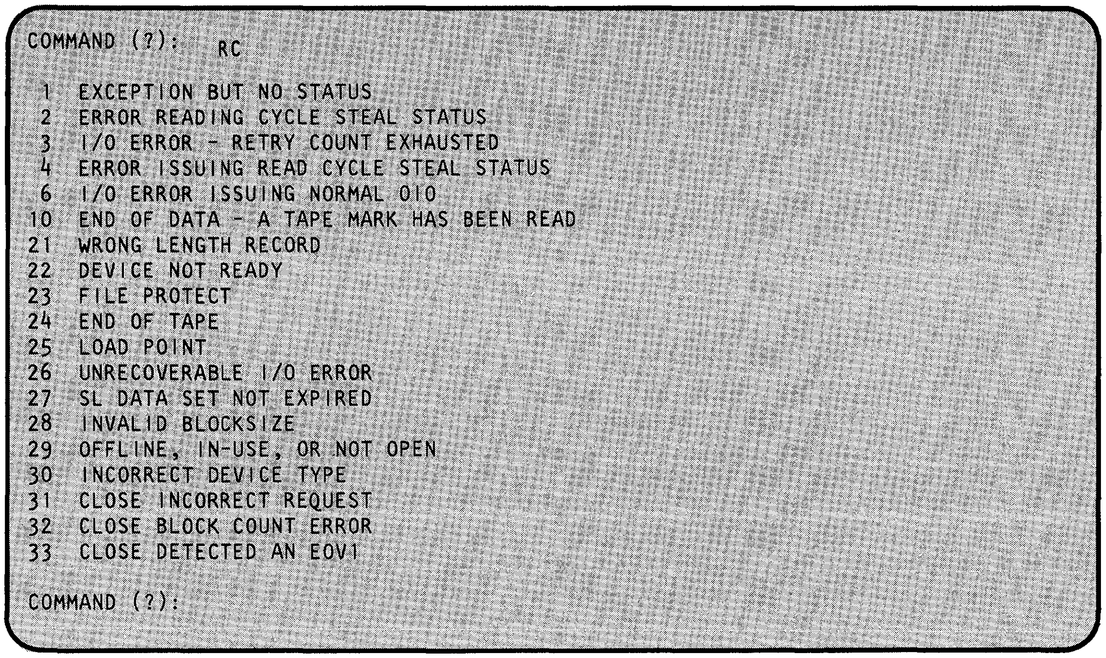
4. The MT command keeps track of the actual record count so you can print this value with the DP command. However, if you issue a forward space file control function (FSF) to move the tape to the end of the file and then issue a back space records control function (BSR) to back up the tape a specified number of records, MT cannot track the actual record count. When this occurs, MT keeps track of the relative record count (relative to the start of the tape) and DP prints this value with an R printed after the number indicating the count is relative.

$TAPEUT1 - Tape Management *(continued)*

RC — Display Tape Return Codes

Use the RC command to display the tape return codes on the terminal where you invoked $TAPEUT1.

Example: Display tape return codes.



```
COMMAND (?):   RC

 1  EXCEPTION BUT NO STATUS
 2  ERROR READING CYCLE STEAL STATUS
 3  I/O ERROR - RETRY COUNT EXHAUSTED
 4  ERROR ISSUING READ CYCLE STEAL STATUS
 6  I/O ERROR ISSUING NORMAL OIO
10  END OF DATA - A TAPE MARK HAS BEEN READ
21  WRONG LENGTH RECORD
22  DEVICE NOT READY
23  FILE PROTECT
24  END OF TAPE
25  LOAD POINT
26  UNRECOVERABLE I/O ERROR
27  SL DATA SET NOT EXPIRED
28  INVALID BLOCKSIZE
29  OFFLINE, IN-USE, OR NOT OPEN
30  INCORRECT DEVICE TYPE
31  CLOSE INCORRECT REQUEST
32  CLOSE BLOCK COUNT ERROR
33  CLOSE DETECTED AN EOV1

COMMAND (?):
```

$TAPEUT1 - Tape Management *(continued)*

RT and ST - Saving and Restoring a Disk Device, Volume, or Data Set

To save a disk device, volume, or data set on tape, use the ST command. To restore a disk device, volume, or data set from tape, use the RT command. Two functions are available that these commands use:

- The first function (called the double-buffered function) uses two buffer areas to hold the output from tape or disk. As the first buffer area is emptying its contents, the utility is filling the second buffer area. By using two buffer areas, the tape drive or disk never has to wait for the system to empty the contents of the buffer area. If the system has sufficient storage available (we recommend a 64K area), it will always use the double-buffered function.

- The second function (called the single-buffered function) uses one buffer area to hold the output from tape or disk. As a result, the tape drive or disk has to wait for the the system to empty the contents of the buffer area to disk.

To help you monitor the save and restore processes, $TAPEUT1 issues a progress report to the terminal where you loaded the utility. The report contains the total number of records the utility must transfer and the number of records that it has transferred already. The system issues the progress report only if you are operating under the double-buffered function.

Example: In this progress report, the utility must transfer a total of 10000 records. The current number of records it has transferred already is 5760.

```
TRANSFER TOTAL=     10000       CURRENT=     5760
```

As the transfer takes place, the system updates the CURRENT= number on your screen until it completes the operation. Once the operation completes, the TOTAL= and CURRENT= numbers are the same.

In addition, when you save or restore tapes using the double-buffered function, you can initialize and vary tapes online automatically.

RT - Restore Disk Device, Volume, or Data Set from Tape

Use the RT command to restore a disk device, volume, or data set from tape. To restore information from a tape, you must have created the tape previously using the ST command in the same version of EDX. In addition, you can restore a disk volume from a tape to the same device type or a different device type, except for the IPL volume. (The system treats the IPL volume similarly to a disk device; you must restore it to the same device type.) To restore an entire disk device from tape, the device type and model number of the source and target disks must match.

$TAPEUT1 - Tape Management *(continued)*

Two restore functions are available to the RT command:

- The first function (called the double-buffered function) uses two buffer areas to hold the output from tape.
- The second function (called the single-buffered function) uses one buffer area to hold the output from tape.

Automatic Varyon: If you choose automatic varyon and $TAPEUT1 can load the double-buffered function, it issues the following messages:

```
VOLUME RESTORE OF MYVOL FROM TAPE TAPE##00,VOLSAV OK? (Y,N):  Y

USE ATTN/CA TO CANCEL THE RESTORE
```

The system automatically varies online each tape used in the restore process.

If it cannot load the double-buffered function, $TAPEUT1 attempts to load the single-buffered function and issues the following messages:

```
INSUFFICIENT MEMORY TO LOAD DOUBLE-BUFFERED UTILITY

LOADING SINGLE-BUFFERED UTILITY
- WARNING - NO DATA STREAM CAPABILITY WITH SINGLE-BUFFERED UTILITY
SAVE/RESTORE TIME MAY BE SIGNIFICANTLY INCREASED
```

Note: $TAPEUT1 issues the warning message only if you are using a 4968 tape unit.

If $TAPEUT1 is successful in loading the single-buffered function, you can proceed with the restore process; however, you must vary each tape online with the $VARYON operator command.

If $TAPEUT1 cannot load the single-buffered function, it issues return code 70, indicating that there is not enough storage to load the function, and issues the COMMAND (?): prompt.

Note: To have sufficient storage available for the double-buffered function, we recommend that you load the $TAPEUT1 utility into a full 64K partition.

$TAPEUT1 - Tape Management *(continued)*

When you enter RT, $TAPEUT1 issues the following message:

```
AUTO VARYON MODE? (Y,N):  Y
```

If you respond **Y,** $TAPEUT1 automatically varies each tape online.

If you respond **N,** you must vary each tape online yourself.

Then enter all the parameters necessary for the restore function. The following example shows the parameters necessary for a volume restore using automatic varyon.

```
ENTER TAPEID (1-6) CHARACTERS):  TAPE02

**********************************
*  WARNING:  TO ENSURE PROPER    *
*  DISK CONTENTS, THE SYSTEM     *
*  SHOULD BE INACTIVE WHILE      *
*  RUNNING THIS UTILITY          *
**********************************

RESTORE OPTIONS:
 1 = DEVICE
 2 = VOLUME
 3 = DATA SET
 4 = EXIT
ENTER RESTORE OPTION:  2

ENTER TARGET DISK VOLUME NAME:  MYVOL

TAPE##00,VOLSAV ONLINE

(The utility varies the source tape online automatically)
```

After you enter the necessary parameters, $TAPEUT1 attempts to load the double-buffered function.

Nonautomatic Varyon Mode: If you choose nonautomatic varyon mode, $TAPEUT1 goes through the same procedure as outlined under the "Automatic Varyon:" on page UT-540. However, you will need to use the $VARYON operator command to vary online each tape used in the restore process.

$TAPEUT1 - Tape Management *(continued)*

Restoring Multiple Tapes: $TAPEUT1 then prompts you to mount any additional tapes it may need to complete the restore. In the following example using the double-buffered utility (automatic varyon mode), $TAPEUT1 prompts you to mount TAPE##01 which is the next tape in a series of two tapes you are restoring.

```
MOUNT TAPE##01,VOLSAV
REPLY Y WHEN TAPE MOUNTED AND TAPE DRIVE ONLINE?  Y

TAPE##01,VOLSAVE ONLINE

(The system varies the source tape online automatically.)
```

If you did not choose automatic varyon mode, use the $VARYON operator command to vary the tape online.

When it has restored the volume, $TAPEUT1 issues the following message and prompts you for another command.

```
VOLUME RESTORED

COMMAND (?):
```

For the 4968 tape unit, once a restore completes successfully, the system rewinds and unloads the source tape automatically. This also occurs at the end of each tape (EOT), after the system writes the trailer record. If the restore is not successful, the system only rewinds the tape.

Note: If you do not want the source tape to be unloaded after a restore, do not use automatic varyon.

ST — Save a Disk Device, Volume, or Data Set on Tape

Use the ST command to save an entire disk device, volume, or data set on tape. ST prompts you to specify whether you are saving a device, volume, or data set. ST is used in conjunction with the restore command (RT) to back up data you wish to protect.

There are two restore functions available to the ST command:

- The first function (called the double-buffered utility) uses two buffer areas to hold the output from disk.

- The second function (called the single-buffered utility) uses one buffer area to hold the output from disk.

$TAPEUT1 - Tape Management *(continued)*

Automatic Initialization: When operating under the double-buffered function, $TAPEUT1 automatically initializes each tape during the save process. This means that you need not stop a save operation and initialize each tape before saving information on it.

When using auto initialization, $TAPEUT1 assigns the tape data set name, volume name, and owner identification. The default owner identification assigned is "SYSTEM." The names the system assigns for standard label tapes are as follows:

Save Type	**Assigned Volume Label**
DATA SET	TAPE##XX,DATSAV
VOLUME	TAPE##XX,VOLSAV
DEVICE	TAPE##XX,DEVSAV

For bypass and nonlabeled tapes, the system assigns only the trailer label as follows:

Type	**Assigned Volume Label**
SAVE	TAPE##XX,TAPEID (TAPE DRIVE ID)
SAVE	TAPE##XX,TAPEID

$TAPEUT1 increments the last two characters of TAPE##XX for each tape required to complete the save process. For example:

```
HEADER FOR FIRST TAPE = TAPE##00
HEADER FOR NEXT TAPE  = TAPE##01 (tape trailer from first tape)
HEADER FOR NEXT TAPE  = TAPE##02 (tape trailer from second tape)
```

You can modify the names automatically assigned (default parameters) only during initialization of the first tape. $TAPEUT1 automatically initializes any subsequent tapes and writes the trailer records with the new default parameters.

After $TAPEUT1 initializes a tape, it issues one of the following messages depending upon the type of save (device, volume, or data set):

```
DEVICE SAVE OF DISK AT ADDRESS 00C0 ONTO TAPE TAPE##00,DEVSAV OK? (Y,N):
VOLUME SAVE OF MYVOL ONTO TAPE TAPE##00,VOLSAV OK? (Y,N):
DATA SET SAVE OF DATA1,EDX002 ONTO TAPE TAPE##00,DATSAV OK? (Y,N):
```

$TAPEUT1 - Tape Management *(continued)*

If the target tape volume label is satisfactory, respond **Y**. If you want to change the label, respond **N**. $TAPEUT1 then prompts you for the new default tape label.

```
CHANGE DEFAULT TAPE VOLUME LABEL? (Y,N):  Y

INPUT NEW TAPE VOLUME LABEL (1-6 CHARACTERS):  NEWVOL

INPUT OWNER IDENTIFICATION (1-10 CHARACTERS):  D27H
TAPE##00,NEWVOL TAPE INITIALIZED

DATA SET SAVE OF DATA1,EDX002 ONTO TAPE TAPE##00,NEWVOL OK? (Y,N):  Y
```

When you enter ST, $TAPEUT1 issues the following message:

```
AUTO INITIALIZATION MODE? (Y,N):
```

If you respond **Y,** $TAPEUT1 initializes any tapes used for the save process.

If you respond **N,** you must initialize each tape with the IT command and vary the tape online using the $VARYON operator command.

Then enter all the parameters necessary for the save function. The example shown is for a device-save using the double-buffered utility (automatic initialization).

```
ENTER TAPEID (1-6) CHARACTERS):  TAPE02

*********************************
*  WARNING:  TO ENSURE PROPER   *
*  DISK CONTENTS, THE SYSTEM    *
*  SHOULD BE INACTIVE WHILE     *
*  RUNNING THIS UTILITY         *
*********************************

SAVE OPTIONS:
 1 = DEVICE
 2 = VOLUME
 3 = DATA SET
 4 = EXIT
ENTER SAVE OPTION:  1
DISK IODA IN HEX(HH):  C0

TAPE EXPIRATION DATE WILL NOT BE CHECKED PRIOR TO INITIALIZATION
CONTINUE (Y,N):  Y

INPUT EXPIRATION DATE (YYDDD):  83003
 TAPE##00,DEVSAV TAPE INITIALIZED
(The source tape is automatically initialized and varied online.)
```

$TAPEUT1 - Tape Management *(continued)*

After you enter the necessary parameters, the $TAPEUT1 attempts to load the double-buffered utility.

If you choose automatic initialization mode and $TAPEUT1 is successful in loading the double-buffered utility, $TAPEUT1 issues the following messages:

```
DEVICE SAVE OF DISK AT ADDRESS 00C0 ONTO TAPE TAPE##00,DEVSAV OK? (Y,N):  Y
USE ATTN/CA TO CANCEL THE SAVE
```

If $TAPEUT1 cannot load the double-buffered utility, it issues the following message and prompts you for another command.

```
INSUFFICIENT MEMORY TO LOAD DOUBLE-BUFFERED UTILITY
COMMAND (?):
```

$TAPEUT1 does not attempt to load the single-buffered utility because the single-buffered utility does not have the automatic initialization capability.

Note: To have sufficient storage available for the double-buffered utility, we recommend that you load the $TAPEUT1 utility into a full 64K partition.

Nonautomatic Initialization: If you choose nonautomatic initialization, you must first initialize each tape with the IT command and then vary each online with the $VARYON operator command.

If $TAPEUT1 can load the double-buffered utility, it issues the the following prompt:

```
VOLUME RESTORE OF MYVOL FROM TAPE TAPE##00,VOLSAV OK? (Y,N):  Y
USE ATTN/CA TO CANCEL THE RESTORE
```

$TAPEUT1 - Tape Management *(continued)*

If $TAPEUT1 cannot load the double-buffered function, it issues a message and attempts to load the single-buffered function.

```
INSUFFICIENT MEMORY TO LOAD DOUBLE-BUFFERED UTILITY
LOADING SINGLE-BUFFERED UTILITY
```

If it can load the single-buffered function, $TAPEUT1 issues the following messages:

```
-WARNING- NO DATA STREAM CAPABILITY WITH SINGLE-BUFFERED UTILITY
SAVE/RESTORE TIME MAY BE SIGNIFICANTLY INCREASED
```

Note: $TAPEUT1 issues the warning message only if you are using a 4968 tape unit.

If $TAPEUT1 cannot load the single-buffered utility, it issues return code 70 (indicating that there is not enough storage to load the utility) then issues the COMMAND (?): prompt.

Note: To have sufficient storage available for the double-buffered utility, we recommend that you load the $TAPEUT1 utility into a full 64K partition.

Save Process Using Multiple Tapes: If the save process requires more than one tape, $TAPEUT1 prompts you to mount any additional tapes that may be necessary to complete the save. In this example (automatic initialization mode), four tapes are used to save the device.

```
REPLY Y WHEN THE NEXT TAPE IS MOUNTED AND TAPE UNIT ONLINE?  Y
 TAPE##01,DEVSAV TAPE INITIALIZED
REPLY Y WHEN THE NEXT TAPE IS MOUNTED AND TAPE UNIT ONLINE?  Y
 TAPE##02,DEVSAV TAPE INITIALIZED
REPLY Y WHEN THE NEXT TAPE IS MOUNTED AND TAPE UNIT ONLINE?  Y
 TAPE##03,DEVSAV TAPE INITIALIZED
```

Once $TAPEUT1 has saved the device, it issues the following message and prompts you for another command.

```
DEVICE SAVED

COMMAND (?):
```

$TAPEUT1 - Tape Management *(continued)*

TA — Allocate a Tape Data Set:

Use the TA command to delete an existing data set and reallocate a null data set, or add a null data set after the last data set on the volume.

Notes:

1. Use this command to place data set labels on previously initialized standard-labeled (SL) tapes; the tape unit must be in the standard label processing mode.
2. The system destroys all the data on the tape following the newly allocated data set.
3. You must vary the tape online to the file number of the data set you are allocating.
4. You must vary the tape online if you want a program to access it.

Example: Allocate data set (DATA2222) on volume 123456.

```
COMMAND (?): TA

TAPEDS (NAME,VOLUME): DATA2222,123456
EXPIRATION DATA (YYDDD): 79001
DATA SET ALLOCATED

COMMAND (?):
```

$TERMUT1 - Change Terminal Parameters

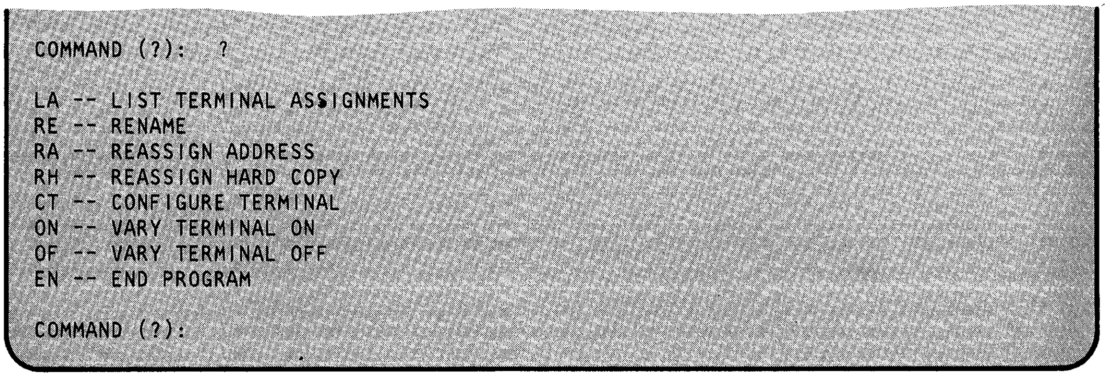
$TERMUT1, a general-purpose terminal utility program, alters logical device names, address assignments, or terminal configuration parameters. Changes you make remain in effect until you reinitialize the system.

Invoking $TERMUT1

Invoke $TERMUT1 with the $L operator command or option 3.10 of the session manager.

$TERMUT1 Commands

To display the $TERMUT1 commands at your terminal, enter a question mark in response to the prompting message COMMAND (?):.



```
COMMAND (?):  ?

LA -- LIST TERMINAL ASSIGNMENTS
RE -- RENAME
RA -- REASSIGN ADDRESS
RH -- REASSIGN HARD COPY
CT -- CONFIGURE TERMINAL
ON -- VARY TERMINAL ON
OF -- VARY TERMINAL OFF
EN -- END PROGRAM

COMMAND (?):
```

After $TERMUT1 displays the commands, it prompts you again with COMMAND (?):. Then you can respond with the command of your choice (for example, LA). Each command and its explanation is presented in alphabetical order on the following pages.

CT — Configure Terminal

Use the CT command to modify the page-formatting parameters associated with a terminal, but be sure you turn on the printer. If you don't, any changes you made will not take effect and $TERMUT1 enters a wait state. You must then turn the printer on and reenter the CT command. With CT, you can also designate a terminal as a printer, temporarily, for spooling output.

CT allows you to do some of the same things as the TERMCTRL function. When you specify the CT command, the system issues questions for the various parameters of the terminal. $TERMUT1 checks the terminal name to see if it is a 4975-01A, 4975, 5219, 5224, or 5225. If the device is one of these printers, the system asks additional questions relating to their extended functions.

$TERMUT1 - Change Terminal Parameters *(continued)*

When you change the CHARSET parameter, it has the same effect as if you had coded it on the TERMINAL statement.

Any parameters changed by the $TERMUT1 CT command become permanent values until you change them again with another CT command or another TERMCTRL statement.

The following examples show the conditional prompt message associated with each parameter for the 4973, 4974, and 4975-01A; for the 3101 (block mode); for the 4975 models 1 and 2, 5219, 5224, and 5225.

Once you assign values for these parameters, you can change them by issuing the CT command again. The CT command displays the values that currently are defined in the NOW IS ' ' portion of the prompt, and you can change or delete the values that you no longer need. If you don't want to change a given value, press the enter key and the system will not change the value.

CT first prompts you for the terminal name. Enter * to specify the terminal where you are assigned at present or enter the name of another terminal you wish to modify.

Example 1: Reconfigure a 4973 or 4974 printer.

```
> $L $TERMUT1
LOADING $TERMUT1    34P,01:05:00, LP= 2600, PART=1

*** TERMINAL CONFIGURATOR ***

COMMAND (?):  CT

ENTER TERMINAL NAME:  $SYSPRTR

PAGE SIZE        (NOW IS 24):
                              VALUE NOT CHANGED
TOP MARGIN       (NOW IS 0):
                              VALUE NOT CHANGED
BOTTOM MARGIN    (NOW IS 23):
                              VALUE NOT CHANGED
HISTORY LINES    (NOW IS 12):  0

LEFT MARGIN      (NOW IS 0):
                              VALUE NOT CHANGED
RIGHT MARGIN     (NOW IS 79):
                              VALUE NOT CHANGED
OVERFLOW LINES   (NOW IS N) :
                              VALUE NOT CHANGED
OUTPUT PAUSE     (NOW IS Y) :
                              VALUE NOT CHANGED
SPOOLABLE (Y/N)  (NOW IS N) :
                              VALUE NOT CHANGED
LPI (6/8)        (NOW IS 8) :  6
```

$TERMUT1 - Change Terminal Parameters *(continued)*

Note: If your terminal is a 4975-01A ASCII printer, the prompts are identical to those in example 1 except for LPI (lines-per-inch). For a 4975-01A ASCII printer, the system does not prompt you for LPI.

If your terminal is a 3101 (block mode), CT prompts you as follows:

Example 2: Reconfigure a 3101 display terminal (block mode).

```
OVERFLOW LINES    (NOW IS N) :
                                VALUE NOT CHANGED
OUTPUT PAUSE      (NOW IS N) :
                                VALUE NOT CHANGED
SPOOLABLE (Y/N)   (NOW IS N) :
                                VALUE NOT CHANGED
ATTRIBUTE (HIGH/LOW/BLINK/BLANK/NO): (NOW IS HIGH)
                                VALUE NOT CHANGED
STREAM (Y/N)?:    (NOW IS N)
                                VALUE NOT CHANGED

COMMAND (?):
```

With the option SPOOLABLE, you can temporarily designate a terminal as a printer for spooling output. CT prompts you as follows with the current status of the terminal displayed.

```
SPOOLABLE (Y/N) (NOW IS __) :
```

If you respond Y, you may specify the designated terminal as a spool device by using the $SPLUT1 utility. If you respond N, the system resets the terminal's status and you may not designate the terminal as a spool device with the $SPLUT1 utility. If you do not want to change the spoolable status of the device, press the enter key and the system will not change the status.

The operation of a terminal you specified as a spoolable device is not affected until you designate the device as a spool device using the $SPLUT1 utility and the system activates spooling. Once the system activates spooling, it dedicates the device to spooling until the spool function ends. For a description of spooling to devices other than IBM printers, see the *Event Driven Executive Language Programming Guide.*

Note: You may get unpredictable results if you change the spoolable status of a terminal while spooling is active on that terminal.

The option OUTPUT PAUSE allows you to disable the "screen full" pause for screen devices so that unattended systems do not enter an indefinite wait state.

$TERMUT1 - Change Terminal Parameters *(continued)*

Note: For more information on terminal parameters, see the TERMINAL statement in the *Installation and System Generation Guide*.

Because the 4975 model 1, 5219, 5224, and 5225 printers do not support text mode, the system does not issue the print mode (PMODE) prompt for these printer models. You may not specify PDEN=COMP for a 4975 model 1 printer.

If your terminal is a 4975 model 1, 5219, 5224, or 5225 printer, CT prompts you as follows:

Example 3: Reconfigure a 4975 model 1, 5219, 5224, or 5225 printer.

```
PDEN  (EXPD/NORM/COMP)  (NOW IS EXPD)  :
                               VALUE NOT CHANGED
CHARSET (? FOR LIST)    (NOW IS USCA)  :
                               VALUE NOT CHANGED

COMMAND (?):
```

If your terminal is a 4975 model 2, CT prompts you as follows:

Example 4: Reconfigure a 4975 model 2 printer.

```
PDEN  (EXPD/NORM/COMP)   (NOW IS EXPD)  :
                               VALUE NOT CHANGED
PMODE (DRAFT/TEXT1/TEXT) (NOW IS TEXT)  :
                               VALUE NOT CHANGED
CHARSET (? FOR LIST)     (NOW IS USCA)   : ?
    INTL    USCA    AUGE   BELG
    BRZL    FRCA    DNNR   SWFI
    FRAN    ITAL    JAEN   KANA
    PORT    SPAN    SPNS   UKIN

CHARSET (? FOR LIST)     (NOW IS USCA)   : INTL

COMMAND (?):
```

$TERMUT1 - Change Terminal Parameters *(continued)*

EN — End $TERMUT1

Use the EN command to end the $TERMUT1 utility.

Example: End $TERMUT1.

```
COMMAND (?):  EN

$TERMUT1 ENDED AT 08:36:02
```

LA — List Terminal Assignments

Use the LA command to list the current terminal names, addresses, types, assigned partitions, hard-copy terminals, and virtual terminal connections, if applicable. An arrow on the left side of the listing identifies the terminal that invoked $TERMUT1.

Example: List terminal assignments.

```
> $L $TERMUT1
LOADING $TERMUT1    34P,01:05:20, LP= 2600, PART=1

*** TERMINAL CONFIGURATOR ***

COMMAND (?):  LA

    NAME      ADDR    TYPE      PART   HARD-COPY  ONLINE

    PRTR1      58     4975-02R    1               YES
    ACCA3101   59     MFA 3101    4               YES (3101 BLOCK MODE)
    PRTR2      5A     4975-01L    1               YES
    PRTR3      5B     4975-02L    1               YES
    $SYSLOG    04     4979        2    MPRINTER   YES
===>$SYSLOGA   24     4978        3    PRTR1      YES
    $SYSPRTR   21     4973 PTR    1               YES
    MPRINTER   01     4974 PTR    1               YES
    ACCA68     68     FPCA 4/L    2               YES (3101 BLOCK MODE)
    CDRVTA     **     VIRT        1               YES CONNECTED CDRVTB
                                                      SYNC=YES
    CDRVTB     **     VIRT        1               YES CONNECTED CDRVTA

COMMAND (?):
```

$TERMUT1 - Change Terminal Parameters *(continued)*

OF — Vary a Terminal Offline

Use the OF command to vary a terminal offline. When you vary a terminal offline, the system ignores any input from that terminal. If you send output to a terminal that is offline, the system issues a return code of 5 "device not ready".

Note: If you vary offline the base address of a 2091/92 card (multiline ACCA) or a 2095/96 card (feature programmable communications adapter), any other addresses serviced by that card become unusable.

Example 1: Vary offline the terminal designated as the alternate logging device.

```
COMMAND (?):  OF
ENTER TERMINAL NAME, OR ADDRESS IN ():  $SYSLOGB

TERMINAL SUCCESSFULLY VARIED OFF

COMMAND (?):
```

If you issue the OF command to a terminal that is in use, CT issues a prompt asking if you wish to continue or end the command. If you choose to continue, normal processing of the program which was using the terminal is not guaranteed.

Example 2: Vary the alternate logging device, which is currently in use, offline.

```
COMMAND (?):  OF
ENTER TERMINAL NAME, OR ADDRESS IN ():  $SYSLOGB

WARNING - TERMINAL IN USE
CONTINUE VARYOFF PROCESSING?  Y

TERMINAL SUCCESSFULLY VARIED OFF

COMMAND (?):
```

$TERMUT1 - Change Terminal Parameters *(continued)*

If you issue the OF command to the terminal that you are using, CT issues a prompt asking if you wish to continue or end the command. If you choose to continue, the utility ends after the VARYOFF completes.

Example 3: Vary offline the terminal you are using.

```
COMMAND (?):  OF
ENTER TERMINAL NAME, OR ADDRESS IN ():  $SYSLOGB

WARNING - TERMINAL IN USE
CONTINUE VARYOFF PROCESSING?  Y

TERMINAL SUCCESSFULLY VARIED OFF

COMMAND(?):
```

ON — Vary a Terminal Online

Use the ON command to vary a terminal online. When the varyon process completes, the terminal can send and receive interrupts.

For any terminal connected to an ACCA-type card, you must send a message to the terminal before it is fully enabled for regular processing. Use the list supervisor configuration command (LS) of the $IOTEST utility to determine which terminals are connected to an ACCA-type card. To send a message to a terminal, use the $TERMUT3 utility.

Example: Vary online the terminal at address 68.

```
COMMAND (?):  ON
ENTER TERMINAL NAME, OR ADDRESS IN ():  (068)

TERMINAL SUCCESSFULLY VARIED ON

COMMAND(?):
```

$TERMUT1 - Change Terminal Parameters *(continued)*

RA — Reassign Address

Use the RA command to reassign the address specified in the address= parameter of the TERMINAL statement. The syntax of this command is:

```
RA  name address
```

where the address in question must be unassigned currently.

As shown in the following examples, the name can be a logical name or hexadecimal device address. If you indicate a device address, it must consist of 1 or 2 digits enclosed in parentheses.

Example:

```
RA   DISPLAY2 7
RA   $SYSPRTR 12
RA   (05) 06
```

RE — Rename Logical Device

Use the RE command to rename the logical terminal name (the label on the TERMINAL statement) that you specified during system generation. The form for this function is:

```
RE  oldname newname
```

The new name replaces the old name. As shown in the following examples, the old name can be a logical name or a hexadecimal device address. If you indicate a device address, it must consist of 1 or 2 digits enclosed in parentheses. Use the LA command to verify the changes.

Example:

```
RE   DISPLAY2 DISPLAY3
RE   (06) TERM1
```

$TERMUT1 - Change Terminal Parameters *(continued)*

RH — Reassign Hard-Copy

Use the RH command to change the hard-copy device associated with a 4978, 4979, or 4980 display station and to indicate which program function key you want to use for the hard copy. The form is:

```
RH  name keycode
```

Here, "name" is the logical name (not device address) of the hard-copy device, and "keycode" is the code for the desired hard-copy key (for example, 1 to 6 for the 4979 display).

The system changes the hard copy device for the terminal from which you loaded $TERMUT1.

Example:

```
RH   $SYSLOGA 6
RH   $SYSLOGA 4
RH   PRTR2 6
```

$TERMUT2 - Change Image/Control Store

You can use $TERMUT2 to:

- Restore the image buffer of a 4974 printer to the standard 64-character set.
- Load the image buffer of a 4974 with the 96-character set $4974IS1. Appendix B of *4974 Printer Description*, GA34-0025, includes a description of the 96-character set.
- Assign a DEFINE key in a 4978 control store.

  Note: The 4980 has a permanent DEFINE key.
- Change the definition of one or more keys in a 4978 or 4980 control store.
- Load a 4978 or 4980 control store from a direct-access data set or save a newly redefined 4978 or 4980 control store into a direct-access data set.
- Load a 4978 or 4980 image store from a direct-access data set or save a 4978 or 4980 image store into a direct-access data set. Refer to the description of the $FONT utility program for a description of image store definition.
- Completely load a 4980 terminal.

You may wish to invoke these functions from a terminal other than the one you are using; therefore, the system prompts you to specify a terminal. If the selected terminal is not a 4974, 4978, or 4980, the system notifies you and rejects the command.

4974 Support

Use $TERMUT2 to restore the image buffer of a 4974 printer to the standard 64-character set or to load the 96-character set image $4974IS1. The 4974 printer uses the Extended Binary Coded Decimal Interchange Code (EBCDIC), which includes 64 standard characters and five characters for international use (96 characters are available with $4974IS1). You can change the standard key definition by using the TERMCTRL instruction within your application program or using the $FONT utility, and the system stores the redefined character set in the image buffer of the 4974. For detailed information on the 4974 printer, refer to the Bibliography for the 4974 Printer manual.

4978 and 4980 Support

Use $TERMUT2 to make special character string assignments on the 4978 and 4980 keyboard. The system defines the functional characteristics of a keyboard by data tables in the system-supplied data sets. These tables vary according to the particular keyboard used and are provided on an IBM diskette shipped with your keyboard. The tables contained on the diskette are of two types:

- Control data
- Image data.

$TERMUT2 - Change Image/Control Store *(continued)*

Control data consists of scan-code-translation tables and a redefine table. Image data consists of a character-image table.

4978 and 4980 displays have a control store and an image store, loaded from disk or diskette. At IPL, the system automatically loads all 4978s and 4980s with the following control-store and image-store data sets:

Display Station	Control Store	Image Store
4978	$4978CS0	$4978IS0
4980	$4980CS0	$4980IS0

These control/image stores may be standard system-supplied data sets or control/image store data sets that you created and named $4978CS0 and $4978IS0 for the 4978 and $4980CS0 and $4980IS0 for the 4980.

Key definitions can be changed, perhaps to be appropriate to a special key data application, and the redefined keyboard definitions saved on disk. The keyboard definition can be reloaded later using $TERMUT2 or by using the TERMCTRL instruction within your application. When the tables are altered, it is not necessary to alter the entire table. Your particular application may be enhanced through minor changes in key functions on a temporary or permanent basis. Use $FONT to change the display image of redefined keys. Refer to the *Operation Guide* for procedures on how to define an interrupt key on a 4978 or 4980.

For detailed information on the 4978 or 4980 display station functions and the 4978 or 4980 keyboards, refer to the Bibliography.

Data Set Names and Requirements

Depending on the type of display stations that are attached to your Series/1, two special names are reserved by the system and used during initial program load time. For the 4978 and 4980 display stations, the two special data sets are as follows:

Display Station	Control Store Label	Image Store Label
4978	$4978CS0	$4978IS0
4980	$4980CS0	$4980IS0

These data sets are automatically searched for and loaded to defined display stations during the initialization phase at IPL time. After IPL, $TERMUT2 can be used to load a control or image store from control/image data sets you have defined, or to read the control or image store in a display, and write to a data set you have allocated.

For each control store or image store that you create and wish to save, $TERMUT2 requires preallocated data sets. For the 4978, the sizes of the data sets are as follows:

1. A 16-record (4096 bytes) data set for each control store
2. An 8-record (2048 bytes) data set for each image store.

$TERMUT2 - Change Image/Control Store *(continued)*

For the 4980, the sizes of the data sets are as follows:

1. An 8-record (2048 bytes) data set for each control store
2. A 16-record (4096 bytes) data set for each image store

Names for the image and control stores respectively are $4980ISx and $4980CSx, where 'x' is any character excluding 0 through 9 which are reserved by EDX.

Invoking $TERMUT2

You invoke $TERMUT2 with the $L operator command or option 4.2 of the session manager.

$TERMUT2 Commands

To display the $TERMUT2 commands at your terminal, enter a question mark in response to the prompting message COMMAND (?):.

```
> $L $TERMUT2
LOADING $TERMUT2   34P,01:07:09, LP= 9200, PART=1

COMMAND (?):  ?
AD - ASSIGN DEFINE KEY
C  - CHANGE KEY DEFINITION
EN - END PROGRAM
LC - LOAD CONTROL STORE
LI - LOAD IMAGE STORE
LT - LOAD 4980 TERMINAL
RE - RESTORE 4974 TO STANDARD 64-CHARACTER SET
SC - SAVE CONTROL STORE
SI - SAVE IMAGE STORE

COMMAND (?):
```

After $TERMUT2 displays the commands, you are again prompted with COMMAND (?):. Then you can respond with the command of your choice (for example, AD).

$TERMUT2 - Change Image/Control Store *(continued)*

AD — Assign a Define Key

Use the AD command to predefine a key that causes the 4978 attachment to enter define mode and allows you to assign a function or a series of functions to a specific key. For example, if you wanted to use key number 66 (A in Figure 35 on page UT-563), find that number in Figure 37 on page UT-564 (A) and make note of its scan code, in this case 20. The 4980 has a predefined DEFINE key that you may want to use instead of the AD command. For a step-by-step explanation of this procedure, refer to "Chapter 3" of the *Operation Guide.*

Note: The 4980 has a predefined DEFINE key.

Syntax:

```
AD       number terminal

Required:  ALL
Default:   none
```

Operands	*Description*
number	Hexadecimal number of the key assigned as the DEFINE key.
terminal	The name of the terminal whose control store you are modifying. Pressing the enter key (CR=carriage return) or entering an * and pressing the enter key specifies the terminal that you used to invoke $TERMUT2.

Example: Assign a define key.

```
COMMAND (?):  AD
  ENTER SCAN CODE OF KEY TO BE ASSIGNED
    AS THE DEFINE KEY (HEX):  20
  ENTER TERMINAL NAME (CR OR * = THIS ONE):   $SYSLOG

  ENTER NONSHIFT, SHIFT, OR ALTERNATIVE CODE MODE (N/S/A):  N
```

Note: The N/S/A options will appear only on the 4980.

$TERMUT2 - Change Image/Control Store *(continued)*

C — Change a Key Definition

Use the C command to change the function ID, local character, or interrupt code of a key to another definition on the terminal you specify. For a step-by-step explanation of this procedure, refer to "Chapter 3" of the *Operation Guide.*

Syntax:

```
C        terminal key id char interrupt

Required: ALL
Default:  none
```

Operands	*Description*
terminal	The name of the terminal whose control store you are modifying. Press the enter key (CR=carriage return) or enter an * and press the enter key to specify the terminal that you used to invoke $TERMUT2.
key	Defines the key (in hexadecimal) you want to redefine.
id	Defines the new function of the key (in hexadecimal).
char	Defines the EBCDIC code (in hexadecimal) used to print the character on the screen.
interrupt	Defines the key (in hexadecimal) as an interrupt key or a character.

Notes:

1. The decimal and hexadecimal values associated with the interrupt key code are reduced by your system. These values decrease by the number specified for the PF1 key during system generation. If you used the default value for PF1 at system generation it decreases by 1, the default. To verify the system code use the $PFMAP utility. Note that values for the interrupt code have been reduced by the value specified for PF1 at system generation.

2. You can find a partial listing of the scan code, function ID code, character/function code, and interrupt code for the 4978, PRPQ D02056 in Figure 37 on page UT-564 and for the 4980 in Figure 38 on page UT-565. For a complete listing, see the general information manuals for the 4978 keyboard or the 4980 display station.

$TERMUT2 - Change Image/Control Store *(continued)*

Example: Change a key definition.

```
COMMAND (?):  C
  ENTER TERMINAL NAME (CR OR * = THIS ONE):  $SYSLOG
  ENTER SCAN CODE OF THE KEY TO BE REDEFINED (HEX):  01
  ENTER NONSHIFT, SHIFT, OR ALTERNATIVE CODE MODE (N/S/A):  N
  ENTER NEW FUNCTION ID (HEX):  20
  ENTER NEW CHARACTER/FUNCTION CODE (HEX):  00
  ENTER NEW INTERRUPT CODE (HEX):  01
  ANOTHER KEY?  N
```

Note: The N/S/A options will appear only on the 4980.

$TERMUT2 - Change Image/Control Store *(continued)*

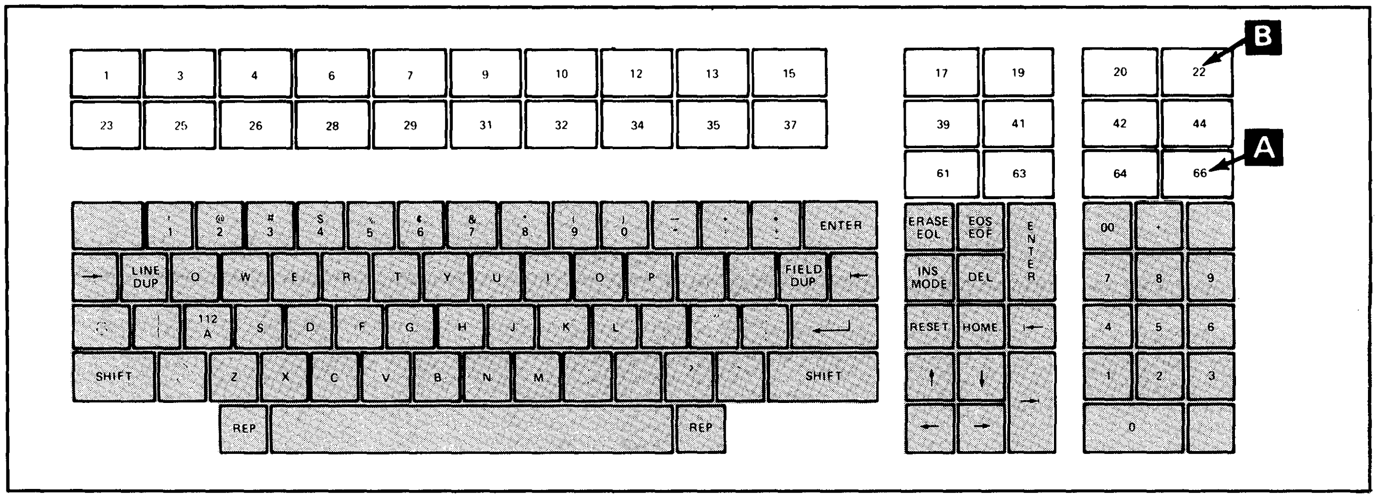


Figure 35. 4978 Display Station keyboard

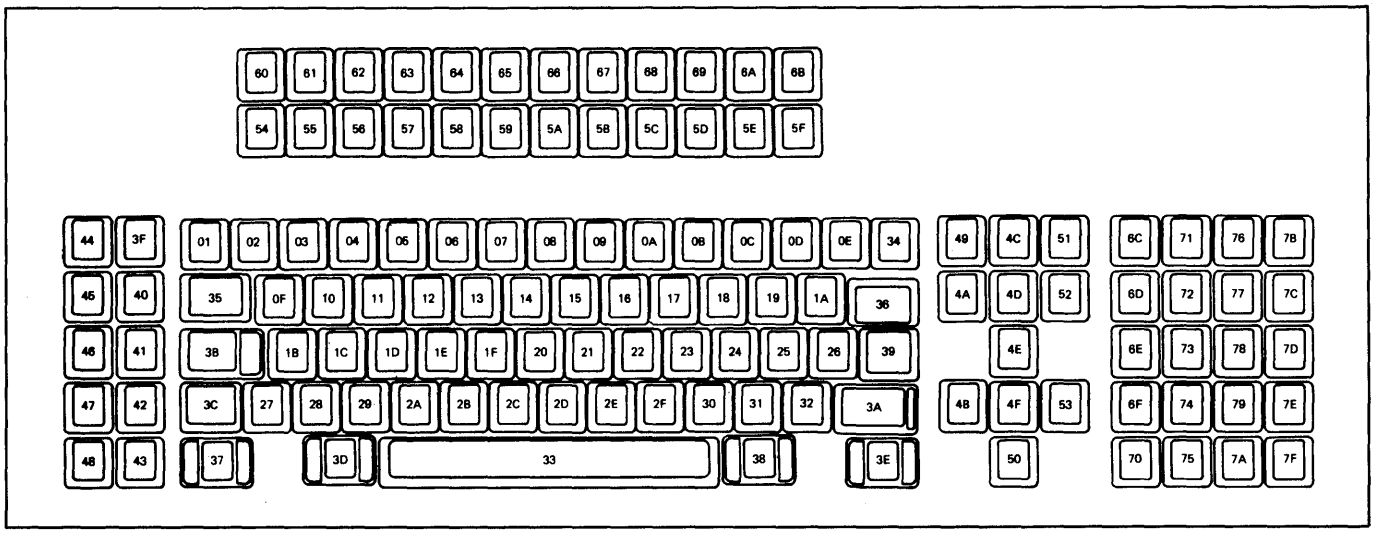


Figure 36. 4980 Display Station

$TERMUT2 - Change Image/Control Store *(continued)*

Control Store Data

Downshift Unshifted

Key position	Scan code	Function ID code	Character/local function code	Interrupt code	Character image table								Keytop symbol
					Row								
					0	1	2	3	4	5	6	7	
1	01	20	00	01									
3	02	20	00	02									
4	03	20	00	03									
6	04	20	00	04									
7	05	20	00	05									
9	06	20	00	06									
10	07	20	00	07									
12	08	20	00	0B									
13	09	20	00	0C									
15	0A	20	00	0D									
17	0B	20	00	0E									
19	0C	20	00	0F									
20	0D	20	00	10									
22	0E	20	00	11									
23	0F	20	00	12									
25	10	20	00	13									
26	11	20	00	14									
28	12	20	00	15									
29	13	20	00	16									
31	14	20	00	17									
32	15	20	00	18									
34	16	20	00	19									
35	17	20	00	1A									
37	18	20	00	1B									
39	19	20	00	1C									
41	1A	20	00	1D									
42	1B	20	00	1E									
44	1C	20	00	1F									
61	1D	20	00	20									
63	1E	20	00	21									
64	1F	20	00	22									
66	20	20	00	23									
67	21	70	00	00	00	00	00	00	00	00	00	00	(Blank)
112	4C	00	C1	00	30	05	48	48	78	48	48	00	A
113	4D	00	E2	00	07	48	04	30	01	48	07	00	S
114	4E	00	C4	00	70	05	0C	0C	0C	05	70	00	D
115	4F	00	C6	00	78	40	40	70	40	40	40	00	F
116	50	00	C7	00	31	0C	40	40	58	0C	38	00	G
117	51	00	C8	00	48	48	48	78	48	48	48	00	H
118	52	00	D1	00	0B	01	01	01	41	41	34	00	J

Upshift – Shifted

Key position	Scan code	Function ID code	Character/local function code	Interrupt code	Character image table								Keytop symbol
					Row								
					0	1	2	3	4	5	6	7	
1	81	20	00	01									
3	82	20	00	02									
4	83	20	00	03									
6	84	20	00	04									
7	85	20	00	05									
9	86	20	00	06									
10	87	20	00	07									
12	88	20	00	0B									
13	89	20	00	0C									
15	8A	20	00	0D									
17	8B	20	00	0E									
19	8C	20	00	0F									
20	8D	20	00	10									
22	8E	20	00	11									
23	8F	20	00	12									
25	90	20	00	13									
26	91	20	00	14									
28	92	20	00	15									
29	93	20	00	16									
31	94	20	00	17									
32	95	20	00	18									
34	96	20	00	19									
35	97	20	00	1A									
37	98	20	00	1B									
39	99	20	00	1C									
41	9A	20	00	1D									
42	9B	20	00	1E									
44	9C	20	00	1F									
61	9D	20	00	20									
63	9E	20	00	21									
64	9F	20	00	22									
66	A0	20	00	23									
67	A1	70	00	00	00	00	00	00	00	00	00	00	(Blank)
112	CC	00	C1	00	30	05	48	48	78	48	48	00	A
113	CD	00	E2	00	07	48	04	30	01	48	07	00	S
114	CE	00	C4	00	70	05	0C	0C	0C	05	70	00	D
115	CF	00	C6	00	78	40	40	70	40	40	40	00	F
116	D0	00	C7	00	31	0C	40	40	58	0C	38	00	G
117	D1	00	C8	00	48	48	48	78	48	48	48	00	H
118	D2	00	D1	00	0B	01	01	01	41	41	34	00	J

Figure 37. Control Chart for 4978 Display Station

$TERMUT2 - Change Image/Control Store *(continued)*

Key Reference		Function ID Entries		Character/Local Entries			Interrupt Entries	Data Key Functions		
Scan Code	Key Num	Alpha-numeric Mode	Alterna-tive Mode	Non-Shift Entry	Alpha-numeric Shift Entry	Alterna-tive Code Entry		Alternative Default is no Operation Non-shift	Alphanumeric Shift	
00		00	00	00	00	00	00			
01	1	10	02	79	A1	00	00			<-
02	2	10	02	F1	-> BB	00	00	1	|	<-
03	3	10	02	F2	7C	00	00	2	@	<-
04	4	10	02	F3	7B	00	00	3	#	<-
05	5	10	02	F4	5B	00	00	4	$	<-
06	6	10	02	F5	6C	00	00	5	%	
07	7	10	02	F6	-> BA	00	00	6	¬	<-
08	8	10	02	F7	50	00	00	7	&	
09	9	10	02	F8	5C	00	00	8	*	
0A	10	10	02	F9	4D	00	00	9	(	
0B	11	10	02	F0	5D	00	00	0	)	
0C	12	10	02	60	6D	00	00	-	_	
0D	13	10	02	7E	4E	00	00	=	+	
0E	14	10	02	8F	90	00	00	±	°	<-
0F	17	10	02	98	D8	00	00	q	Q	
10	18	10	02	A6	E6	00	00	w	W	
11	19	10	02	85	C5	00	00	e	E	
12	20	10	02	99	D9	00	00	r	R	
13	21	10	02	A3	E3	00	00	t	T	
14	22	10	02	A8	E8	00	00	y	Y	
15	23	10	02	A4	E4	00	00	u	U	
16	24	10	02	89	C9	00	00	i	I	
17	25	10	02	96	D6	00	00	o	O	
18	26	10	02	97	D7	00	00	p	P	
19	27	10	02	-> B0	-> 4F	00	00	¢	!	<-
1A	28	10	02	E0	6A	00	00	\		<-
1B	31	10	02	81	C1	00	00	a	A	
1C	32	10	02	A2	E2	00	00	s	S	
1D	33	10	02	84	C4	00	00	d	D	
1E	34	10	02	86	C6	00	00	f	F	
1F	35	10	02	87	C7	00	00	g	G	
20	36	10	02	88	C8	00	00	h	H	
6C	90	61	02	00	00	00	1D	Replaceable legend key		
6D	91	10	02	F7	F7	00	00	7		
6E	92	10	02	F4	F4	00	00	4		
6F	93	10	02	F1	F1	00	00	1		
70	94	20	02	17	17	00	00	00		
71	95	61	02	00	00	00	1E	Replaceable legend key		
72	96	10	02	F8	F8	00	00	8		
73	97	10	02	F5	F5	00	00	5		
74	98	10	02	F2	F2	00	00	2		
75	99	10	02	F0	F0	00	00	0		
76	100	61	02	00	00	00	1F	Replaceable legend key		
77	101	10	02	F9	F9	00	00	9		
78	102	10	02	F6	F6	00	00	6		
79	103	10	02	F3	F3	00	00	3		
7A	104	10	02	4B	4B	00	00	.		
7B	105	61	02	00	00	00	20	Replaceable legend key		
7C	106	01	02	00	00	00	21	Replaceable legend key		
7D	107	01	02	00	00	00	22	Replaceable legend key		
7E	108	10	02	60	60	00	00	-		
7F	109	10	02	4E	4E	00	00	+		

Figure 38. Control Chart for 4980 Display Station

$TERMUT2 - Change Image/Control Store *(continued)*

EN — End Program

Use the EN command to terminate the $TERMUT2 utility.

Syntax:

```
EN

Required: none
Default:  none
```

Operands ***Description***

None

Example: End the $TERMUT2 utility.

```
COMMAND (?): EN
$TERMUT2 ENDED
```

$TERMUT2 - Change Image/Control Store *(continued)*

LC — Load a Control Store

Use the LC command to load the control-store data set into the terminal specified.

Syntax:

```
LC        name,volume terminal

Required: ALL
Default:  none
```

Operands	*Description*
name	Defines the name of the data set to be accessed.
volume	Defines the name of the volume containing the data set.
terminal	Defines the name of the terminal that loaded the data set; press the enter key (CR=carriage return) or enter an * and press the enter key to specify the terminal that invoked $TERMUT2.

Example: Load a 4978 control store.

```
COMMAND (?): LC
  FROM DATA SET (NAME,VOLUME): $4978CS0,EDX002
  ENTER TERMINAL NAME (CR OR * = THIS ONE): *
```

$TERMUT2 - Change Image/Control Store *(continued)*

LI — Load an Image Store

Use the LI command to load an image store (character image table) on the terminal specified.

Syntax:

```
LI      name,volume terminal

Required: ALL
Default:  none
```

Operands	*Description*
name	Defines the name of the data to be accessed.
volume	Defines the name of the volume containing the data set.
terminal	Defines the name of the terminal that loaded the data set; press the enter key (CR=carriage return) or enter an * and press the enter key to specify the terminal that invoked $TERMUT2.

Example 1: Load a 4978 image store.

```
COMMAND (?): LI
  FROM DATA SET (NAME,VOLUME): $4978IS0,EDX002
  ENTER TERMINAL NAME (CR OR * = THIS ONE): *
```

Example 2: Load a 4974 image store.

```
COMMAND (?):  LI
  FROM DATA SET (NAME,VOLUME):  $4974IS1,EDX002
  ENTER TERMINAL NAME (CR OR * = THIS ONE):  MPRINTER
```

$TERMUT2 - Change Image/Control Store *(continued)*

LT — Load 4980 Terminal

You can use the LT command to complete loading of a 4980 terminal. If you did not include the power-on support for the 4980 and your 4980 gets powered off, you can bring it back up again by using the LT command.

Syntax:

```
LT      terminal

Required: terminal
Default:  none
```

Operands	*Description*
terminal	Defines the 4980 you want to load.

Example: Load a 4980 terminal.

```
COMMAND (?):  LT
  ENTER TERMINAL NAME (CR OR * = THIS ONE):  TERM1

CURRENT IMAGE STORE DATA SET NAME IS $4980ISX
SYSTEM DEFAULT DATA SET IS $4980ISO
ENTER LAST CHARACTER OF DATA SET NAME TO BE USED:  O

CURRENT CONTROL STORE DATA SET NAME IS $4980CSX
SYSTEM DEFAULT DATA SET NAME IS $4980CSO
ENTER LAST CHARACTER OF DATA SET NAME TO BE USED:  X

LOADING IMAGE STORE AND CONTROL STORE TO TERM1

SUCCESSFUL COMPLETION OF 4980 LOAD

COMMAND (?):
```

$TERMUT2 - Change Image/Control Store *(continued)*

RE — Restore 4974 to Standard 64-Character Set

Use the RE command to restore the image buffer of a 4974 printer to the standard 64-character set.

Syntax:

```
RE      terminal

Required: terminal
Default:  none
```

Operands	***Description***
terminal	Defines the device to be restored.

Example: Restore 4974 to standard 64-character set.

```
COMMAND (?): RE
ENTER TERMINAL NAME (CR OR * = THIS ONE): MPRINTER
```

$TERMUT2 - Change Image/Control Store *(continued)*

SC — Save a Control Store

Use the SC command to save a redefined control store in the data set specified.

Syntax:

```
SC       name,volume terminal

Required: ALL
Default:  none
```

Operands	*Description*
name	Defines the name of the data set to be saved.
volume	Defines the name of the volume where the data set is to be saved.
terminal	Defines the name of the terminal issuing the save; press the enter key (CR=carriage return) or enter an * and press the enter key to specify the terminal that invoked $TERMUT2.

Example: Save a control store.

```
COMMAND (?): SC
  SAVE DATA SET (NAME,VOLUME): $4978CS0,EDX002
  ENTER TERMINAL NAME (CR OR * = THIS ONE): $SYSLOG
```

$TERMUT2 - Change Image/Control Store *(continued)*

SI — Save an Image Store

The SI command saves a redefined image store in the data set specified.

Syntax:

```
SI       name,volume terminal

Required: ALL
Default:  none
```

Operands	*Description*
name	Defines the name of the data set being saved.
volume	Defines the name of the volume where the data set is to be saved.
terminal	Defines the name of the terminal issuing the save; press the enter key (CR=carriage return) or enter an * and press the enter key to specify the terminal that invoked $TERMUT2.

Example: Save an image store.

```
COMMAND (?): SI
  SAVE DATA SET (NAME,VOLUME): $4978ISO,EDX002
  ENTER TERMINAL NAME (CR OR *=THIS ONE): $SYSLOG
```

$TERMUT3 - Send Message to a Terminal

$TERMUT3 sends a single-line message from your terminal to any other terminal defined in the system. One message line is sent at a time and you are prompted for the message and the terminal where the message is to be sent, and if additional messages are to be sent.

Invoking $TERMUT3

Invoke $TERMUT3 with the $L operator command or option 4.3 of the session manager. $TERMUT3 does not have commands and only issues prompting messages as follows:

```
> $L $TERMUT3
LOADING $TERMUT3     4P,01:09:40, LP= 9200, PART=1

TERMINAL BROADCAST UTILITY

ENTER TERMINAL NAME:
(Label of the terminal to which message is to go.)

ENTER MESSAGE TO SEND
(Type in message and press the ENTER (CR) key.
The message is transmitted to the destination terminal.)

ANOTHER MESSAGE (Y/N)?
(If yes, you are prompted to enter another message line.)

ANOTHER TERMINAL (Y/N)?
(If yes, select another terminal and repeat the above process.)

DETACH PROGRAM, (WAIT FOR 'ATTN SEND') (Y/N)?
(If yes, the program remains loaded and inactive.
Pressing the ATTN key and entering 'SEND' starts
the above process again.)
```

$TERMUT3 - Send Message to a Terminal *(continued)*

Example 1: Send a message to terminal $SYSLOGA.

```
> $L $TERMUT3

LOADING $TERMUT3       3P,00:41:10, LP= 8800

TERMINAL BROADCAST UTILITY

ENTER TERMINAL NAME:  $SYSLOGA
ENTER MESSAGE TO SEND
THIS IS A TEST MESSAGE.

ANOTHER MESSAGE (Y/N)?  Y
ENTER MESSAGE TO SEND
THIS IS ANOTHER TEST MESSAGE.

ANOTHER MESSAGE (Y/N)?  N

ANOTHER TERMINAL (Y/N)?  N

DETACH PROGRAM, (WAIT FOR 'ATTN SEND') (Y/N)?  N

$TERMUT3 ENDED AT 00:43:05
```

Example 2: Select another terminal after sending a message.

```
       .
       .
       .
ANOTHER MESSAGE (Y,N)?  N

ANOTHER TERMINAL (Y/N)?  Y
ENTER TERMINAL NAME:  $SYSPRTR
ENTER MESSAGE TO SEND
HELLO $SYSPRTR

ANOTHER MESSAGE (Y/N)?  N

ANOTHER TERMINAL (Y/N)?  N

DETACH PROGRAM, (WAIT FOR 'ATTN SEND') (Y/N)?  N

$TERMUT3 ENDED AT 00:54:00
```

$TERMUT3 - Send Message to a Terminal *(continued)*

Example 3: Keep $TERMUT3 active, send a message later.

```
        .
        .
        .
ANOTHER MESSAGE (Y/N)?  N

ANOTHER TERMINAL (Y/N)?  N

DETACH PROGRAM, ( WAIT FOR 'ATTN SEND') (Y/N)?  Y

> SEND
ENTER TERMINAL NAME:  TTY30
ENTER MESSAGE TO SEND
TTY30 - ARE YOU THERE

ANOTHER MESSAGE (Y/N)?  N

ANOTHER TERMINAL (Y/N)?  N

DETACH PROGRAM, ( WAIT FOR 'ATTN SEND') (Y/N)?  Y
```

$TRACEIO - ACCA/EXIO Trace Facility

The $TRACEIO utility will trace the activities of one device attached through an ACCA or EXIO attachment. $TRACEIO is only capable of tracing the activities on a device (one address). Therefore, you must invoke the $TRACEIO utility for each device you want to trace. Multiple traces require multiple loads, one for each address.

$TRACEIO uses an internal buffer (storage area) to record events as they occur. You can increase or decrease the size of the buffer by using the set storage (SS) command of $DISKUT2.

Invoking $TRACEIO

You invoke $TRACEIO with the $L operator command. The session manager does not support this utility.

```
> $L $TRACEIO
LOADING $TRACEIO    55P,03:29:58, LP= 0000, PART= 2

COMMAND (?):
```

$TRACEIO Commands

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

```
COMMAND (?):  ?

DI - DISPLAY TRACE DATA SET
DU - DUMP TRACE BUFFER
EN - END TRACEIO PROGRAM
RE - REPEAT PREVIOUS TRACE
TR - TRACE COMMUNICATIONS LINE

COMMAND (?):
```

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

To use $TRACEIO, you should start out with the TR command and set up the trace for a specified communications line.

Each command and its explanation is presented in alphabetical order on the following pages.

$TRACEIO - ACCA/EXIO Trace Facility *(continued)*

DI — Display Trace Data Set

Use the DI command to display the contents of the trace data set on the device specified. You have the option of monitoring IO instruction executions (OIO) and/or interrupts. You can display the contents on the terminal where you are working currently or direct the contents to any device you specify.

Example: Display contents of the trace data set on MYPRTR.

```
COMMAND (?): DI

TRACE DATA SET (NAME,VOLUME): $TRACEDS,EDX002
DISPLAY OIO EXECUTIONS (Y/N)? Y
DISPLAY INTERRUPTS (Y/N)? Y
ENTER PRINTER NAME: $MYPRTR
ARE ALL PARAMETERS CORRECT (Y/N)? Y
```

DU — Dump Trace Buffer

Use the DU command to dump the contents of the trace buffer. You have the option of monitoring IO instruction executions (OIO) and/or interrupts. You can also display trace progress data on a terminal or printer and/or log to a disk(ette) data set previously allocated using the allocate (AL) command of the $DISKUT1 utility.

$TRACEIO prompts you to determine what you want to do with the trace data set in the $TRACEIO data buffer. You can display, log, or print the data to disk(ette), or you can do any combination of the three.

Example: In this example, the contents of the $TRACEIO data buffer are to be printed on the system printer and also saved in a data set named $TRACEDS.

```
COMMAND (?): DU

DISPLAY OIO EXECUTIONS (Y/N)? Y
DISPLAY INTERRUPTS (Y/N)? Y
PRINT TRACE DATA? Y
ENTER PRINTER NAME: $SYSPRTR
LOG TO DISK(ETTE) (Y/N)? Y
ENTER TRACE DATA SET NAME,VOLUME: $TRACEDS,EDX002

ARE ALL PARAMETERS CORRECT (Y/N)? Y
```

Sample output displayed on the monitoring terminal/printer would appear as in following example.

$TRACEIO - ACCA/EXIO Trace Facility *(continued)*

```
$TRACEIO MONITOR OF ACCA LINE AT ADDRESS -- 08
TRACE DATA SET --------------------------- $TRACEDS,EDX002

EXECUTE RESET 010  CC=7
EXECUTE START 010  CC=2
EXECUTE START 010  CC=2
EXECUTE START 010  CC=7
INTERRUPT          ICC=2  (EXCEPTION)
                   ISB=A0
                          DEVICE-DEPENDENT STATUS AVAILABLE
                          WRONG LENGTH RECORD

                   DCB
                       CONTROL WORD  2004  (INPUT RECEIVED)
                       TIMER 1       0002
                       TIMER 2       000A
                       CHAIN ADDR    0000
                       BYTE COUNT    0008
                       DATA ADDR     1A4E
EXECUTE SCSS 010   CC=7
INTERRUPT          ICC=3  (DEVICE END)
                   STATUS WORD  0    1A54   RESIDUAL ADDR
                                1    0000   FLAG WORD 1
                                2    F400   FLAG WORD 2
                                     DTR    ON
                                     DSR    ON
                                     RTS    ON
                                     CTS    ON
                                     RECEIVE MODE
```

For ACCA devices, status words are interpreted via messages derived from the bit-significant data contained in the CSS words 1 and 2. EXIO status words may also be displayed/printed/logged. However, with EXIO devices, the hardware type is unknown to the software. You must be very familiar with the device in order to use the EXIO support. Therefore, you must interpret the results of the EXIO CSS words.

EN — End $TRACEIO Utility

The EN command terminates the $TRACEIO utility.

Example: End $TRACEIO utility.

```
COMMAND (?):  EN
$TRACEIO ENDED AT 00:00:00
```

$TRACEIO - ACCA/EXIO Trace Facility *(continued)*

RE — Repeat Previous Trace

Use the RE command to repeat the trace previously performed on a communications line.

Example:

```
COMMAND (?): RE
```

TR — Trace Communications Line

You can use the TR command to trace the activities on a specific communications line.

When you enter TR, $TRACEIO prompts you for the address of the line to be traced.

```
COMMAND (?):  TR

ENTER HEX ADDRESS OF LINE TO BE TRACED:
```

If the device at the specified address is not traceable, $TRACEIO issues the following message:

```
TRACE OPTION NOT SELECTED AT SYSGEN
```

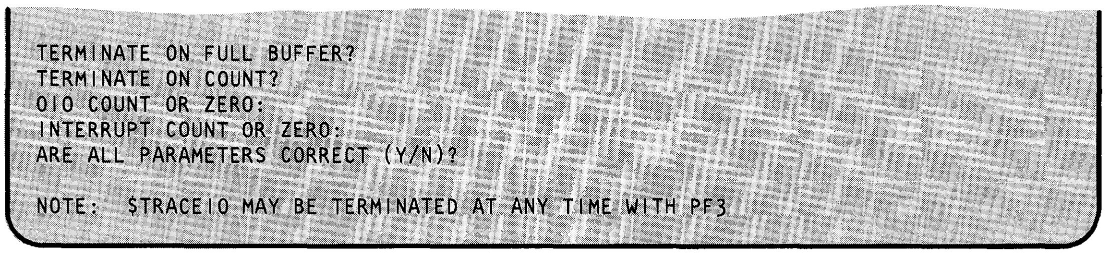
If the device at the specified address is not an ACCA or EXIO device (defined with the TERMINAL or EXIO definition statements, respectively), $TRACEIO issues the following message:

```
INVALID ADDRESS

RETRY (Y/N)?
```

$TRACEIO - ACCA/EXIO Trace Facility *(continued)*

If the device is valid, $TRACEIO automatically adjusts itself to the type of device handler to which it is attached. As a result, you are not required to specify the device type. After verifying that the device is on a traceable line, $TRACEIO modifies the CCB/DDB to point to the immediate action address (FLIH) in the INCLUDE module TRACEIO. $TRACEIO then prompts you to specify one or a combination of the following terminating condition(s):

- OIO count
- interrupt count
- end-of-buffer
- a specific PF key
- a combination of these conditions



```
TERMINATE ON FULL BUFFER?
TERMINATE ON COUNT?
OIO COUNT OR ZERO:
INTERRUPT COUNT OR ZERO:
ARE ALL PARAMETERS CORRECT (Y/N)?

NOTE:  $TRACEIO MAY BE TERMINATED AT ANY TIME WITH PF3
```

If you do not specify end-of-buffer as a terminating condition, the trace buffer wraps on a full-buffer condition as necessary until a specified termination condition (count or PF key) occurs. The first occurrence of any specified termination condition ends the trace regardless of whether other specified criteria have been met.

Interrupts and their condition codes, ISB, and DCB information are recorded. Attempts to execute I/O instructions and their resulting immediate conditions are also recorded. When a device end interrupt from a start cycle steal status (SCSS) instruction is received, the status words are records.

$TRANS - Transmit Data Sets Across a Bisync Line

Load $TRANS in order to transmit EDX data sets and programs from one Series/1 to another across a point-to-point bisynchronous communication line.

Invoking $TRANS

Invoke $TRANS first on the receive end with the $L command. Take this precaution whenever using $TRANS. It will insure that the receive end is ready for any data coming across the line. Only after the receive end has been prepared, invoke $TRANS on the transmit end using the $L command. The session manager does not support this utility. Refer to the *Operation Guide* for step-by-step information on on use of the $TRANS utility.

Load $TRANS on the receiving system. Respond to prompts as shown:

```
> $L $TRANS
LOADING $TRANS     XXP,12:12:12, LP= 0000, PART= 2

$TRANS  -  BISYNC TRANSPORT UTILITY

THIS UTILITY SHOULD ONLY BE USED WHEN TRANSPORTING DATA
SETS OR PROGRAMS OVER A BISYNC LINE.  DO YOU WISH TO CONTINUE (Y/N)? Y

IS THIS THE TRANSMITTING SYSTEM (Y/N)? N

BSC RECEIVE PROGRAM ACTIVE

ENTER BSC LINE ADDRESS IN HEX: 09

BSC LINE OPEN - READY FOR TRANSMISSION
```

$TRANS - Transmit Data Sets Across a Bisync Line *(continued)*

Load $TRANS on the transmitting system. Respond to prompts as shown:

```
> $L $TRANS
LOADING $TRANS     XXP,12:12:12, LP= 0000, PART= 2

$TRANS  -  BISYNC TRANSPORT UTILITY

THIS UTILITY SHOULD ONLY BE USED WHEN TRANSPORTING DATA
SETS OR PROGRAMS OVER A BISYNC LINE.  DO YOU WISH TO CONTINUE (Y/N)? Y

IS THIS THE TRANSMITTING SYSTEM (Y/N)? Y

BSC TRANSMIT PROGRAM ACTIVE

ENTER BSC LINE ADDRESS IN HEX: 09

ENTER SOURCE VOLUME: TEST1

ENTER TARGET VOLUME: TEST2

MEMBERS ON VOLUME TEST1   WILL BE COPIED TO VOLUME TEST2
```

Notes:

1. Messages regarding errors encountered on the receive end will appear on both the receive and transmit ends.

2. If $TRANS is renamed to $INITIAL, it will automatically be loaded at IPL time as a receive end with a default line address of 9. the RECEIVE program will automatically be loaded.

$TRANS Commands

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

```
COMMAND (?): ?
CALL - COPY ALL MEMBERS IN THE VOLUME
CG   - COPY GENERIC
CM   - COPY MEMBER
CV   - CHANGE VOLUME
EN   - END UTILITY
ES   - END UTILITY WITH SHUTDOWN ON RECEIVE END
```

Each command follows in alphabetical order with an explanation.

$TRANS - Transmit Data Sets Across a Bisync Line *(continued)*

CALL — Copy All Members in a Volume from Source to Target

Use CALL to copy all members in a volume or all members starting from a certain member.

Example: Copy all members.

```
COMMAND (?): CALL

COPY FROM THE BEGINNING (Y/N)? Y

ALL MEMBERS ON VOLUME    ZZZ WILL BE COPIED TO VOLUME      AAA
ARE ALL PARAMETERS CORRECT (Y/N)? Y

    ($LOADER and nucleus modules ($EDXNUCx) cannot be copied using
    the CALL command.  Use the CM command to copy these modules.)

$4978CSO   COPY COMPLETE
$LOADER    NOT COPIED
DATA002    COPY COMPLETE

COMMAND (?):
```

Example: Copy all members starting from a certain member.

```
COMMAND (?): CALL

COPY FROM THE BEGINNING (Y/N)? N
ENTER STARTING MEMBER NAME: EDXSVCX

    ($LOADER and nucleus modules ($EDXNUCx) cannot be copied using
    the CALL command.  Use the CM command to copy these modules.)

ARE ALL PARAMETERS CORRECT (Y/N)? Y
EDXSVCX   COPY COMPLETE
#XMTSRC   COPY COMPLETE
$TAPERT   COPY COMPLETE
DISKIO    COPY COMPLETE
RW001     COPY COMPLETE
DATA003    COPY COMPLETE

VOLUME TRANSMISSION COMPLETE
COMMAND (?):
```

$TRANS - Transmit Data Sets Across a Bisync Line *(continued)*

CG — Copy All Members from Source to Target

Use CG to copy all members from one system to another.

Example: Copy All Members from Source to Target.

```
COMMAND (?): CG
ENTER GENERIC TEXT: A

ASMWORK  COPY COMPLETE
ASMOBJ   COPY COMPLETE

COMMAND (?): CG
ENTER GENERIC TEXT: B

NO MATCH ON B

COMMAND (?):
```

CM — Copy Member from Source to Target

Use CM to copy a single member from one system to another.

Example: Copy a member.

```
COMMAND (?): CM

ENTER THE DATA SET NAME: XXX

     (The following prompt appears only when $LOADER
     is specified as the data set you wish to copy.
     Respond "Y" to the prompt issued and the copy will proceed.
     Respond "N" and the utility will return to command mode.)

COPYING $LOADER TO AN IPL VOLUME MAY REQUIRE YOU TO RE-IPL THE SYSTEM.
DO YOU WISH TO CONTINUE (Y/N)? Y

XXX      COPY COMPLETE

COMMAND (?):
```

$TRANS - Transmit Data Sets Across a Bisync Line *(continued)*

CV — Change Source Volume

Use CV to change source volume.

Example: Change the source volume.

```
COMMAND (?): CV
ENTER SOURCE VOLUME: ZZZ
ENTER TARGET VOLUME: AAA
MEMBERS ON VOLUME    ZZZ WILL BE COPIED TO VOLUME      AAA

COMMAND (?):
```

EN — End $TRANS

Use EN to end $TRANS at the transmit end.

Example: End the $TRANS utility.

```
COMMAND (?): EN

BSC LINE CLOSED
$TRANS     ENDED AT 12:14:00
```

ES — End $TRANS Utility with Shutdown on Receive End

Use ES to end $TRANS at the transmit and receive ends.

Example: End the $TRANS utility with shutdown on the transmit and receive ends.

```
COMMAND (?): ES

BSC LINE CLOSED
REMOTE BSC LINE CLOSED
$TRANS     ENDED AT 12:14:00
```

$TRANS - Transmit Data Sets Across a Bisync Line *(continued)*

Example 1: The following example illustrates use of the CV (Change Volume,) CM (Copy Member) and CG (Copy Generic) commands.

```
> $L $TRANS
LOADING $TRANS     XXP,12:12:12, LP= 0000, PART= 2

$TRANS  -  BISYNC TRANSPORT UTILITY

THIS UTILITY SHOULD ONLY BE USED WHEN TRANSPORTING DATA SETS
OR PROGRAMS OVER A BISYNC LINE.  DO YOU WISH TO CONTINUE (Y/N)? Y

IS THIS THE TRANSMITTING SYSTEM (Y/N)? Y

BSC TRANSMIT PROGRAM ACTIVE

ENTER BSC LINE ADDRESS IN HEX: 60

ENTER SOURCE VOLUME: ZZZ

ENTER TARGET VOLUME: AAA

MEMBERS ON VOLUME ZZZ    WILL BE COPIED TO VOLUME AAA
COMMAND (?): ?
CALL - COPY ALL MEMBERS THE VOLUME
CG   - COPY GENERIC
CM   - COPY MEMBER
CV   - CHANGE VOLUME
EN   - END UTILITY
ES   - END UTILITY WITH SHUTDOWN ON RECEIVE END

COMMAND (?): CM
ENTER THE DATA SET NAME: XXX

XXX      COPY COMPLETE

COMMAND (?): CG
ENTER GENERIC TEXT: A

ASMWORK  COPY COMPLETE
ASMOBJ   COPY COMPLETE

COMMAND (?): CG
ENTER GENERIC TEXT: B

NO MATCH ON B

COMMAND (?): EN

BSC LINE CLOSED
$TRANS      ENDED AT 12:14:00
```

$TRANS - Transmit Data Sets Across a Bisync Line *(continued)*

Example 2: The following example illustrates use of the CALL (Copy All Members) command.

```
> $L $TRANS
LOADING $TRANS     XXP,12:12:12, LP= 0000, PART= 2

$TRANS  -  BISYNC TRANSPORT UTILITY

THIS UTILITY SHOULD ONLY BE USED WHEN TRANSPORTING DATA SETS
OR PROGRAMS OVER A BISYNC LINE.  DO YOU WISH TO CONTINUE (Y/N)? Y

IS THIS THE TRANSMITTING SYSTEM (Y/N)? Y

BSC TRANSMIT PROGRAM ACTIVE

ENTER BSC LINE ADDRESS IN HEX: 60

ENTER SOURCE VOLUME: ZZZ

ENTER TARGET VOLUME: AAA

MEMBERS ON VOLUME ZZZ   WILL BE COPIED TO VOLUME AAA
COMMAND (?): CALL

COPY FROM THE BEGINNING (Y/N)? Y

ALL MEMBERS ON VOLUME    ZZZ WILL BE COPIED TO VOLUME      AAA
ARE ALL PARAMETERS CORRECT (Y/N)? Y

ASMOBJ   COPY COMPLETE
ASMWORK  COPY COMPLETE
BACKUP   COPY COMPLETE
LINKWORK COPY COMPLETE

VOLUME TRANSMISSION COMPLETED

COMMAND (?): EN

BSC LINE CLOSED
$TRANS      ENDED AT 12:14:00
```

$TRANS - Transmit Data Sets Across a Bisync Line *(continued)*

Example 3: This example illustrates use of the CALL (Copy All Members) command that runs into overflow problems attempting to complete the copy.

```
> $L $TRANS
LOADING $TRANS     XXP,12:12:12, LP= 0000, PART= 2

$TRANS  -  BISYNC TRANSPORT UTILITY

THIS UTILITY SHOULD ONLY BE USED WHEN TRANSPORTING DATA SETS
OR PROGRAMS OVER A BISYNC LINE.  DO YOU WISH TO CONTINUE (Y/N)? Y

IS THIS THE TRANSMITTING SYSTEM? Y

BSC TRANSMIT PROGRAM ACTIVE

ENTER BSC LINE ADDRESS IN HEX: 60

ENTER SOURCE VOLUME: ZZZ

ENTER TARGET VOLUME: AAA

MEMBERS ON VOLUME TEST   WILL BE COPIED TO VOLUME TEST

COMMAND (?): CALL
COPY FROM THE BEGINNING (Y/N)? Y

ALL MEMBERS ON VOLUME    ZZZ WILL BE COPIED TO VOLUME      AAA

ARE ALL PARAMETERS CORRECT (Y/N)? Y

ASMOBJ   COPY COMPLETE
ASMWORK  COPY COMPLETE
BACKUP   COPY COMPLETE
LINKWORK COPY COMPLETE

DIRECTORY OR VOLUME OVERFLOW ON REMOTE SYSTEM
CONTINUE ON ANOTHER VOLUME? Y

ENTER CONTINUATION VOLUME NAME: BBB

LINKWRK2 COPY COMPLETE
LINKWRK3 COPY COMPLETE

VOLUME TRANSMISSION COMPLETED

COMMAND (?): EN

BSC LINE CLOSED
$TRANS      ENDED AT 12:14:00
```

$TRAP - Save Storage on Error Condition

The $TRAP utility enables you to isolate intermittent hardware and software problems When an error condition occurs, the utility writes, or "dumps," the contents of the hardware registers and processor storage to a disk or diskette data set.

You can set $TRAP to dump processor storage when a specific type of error occurs, such as a program check. If the expected error or failure does not occur, you can force the utility to dump the contents of processor storage by entering the TRAPDUMP command or by pressing the console interrupt button on the programmer console. The use of the TRAPDUMP command and the console interrupt button are explained under "Forcing a $TRAP Dump" on page UT-593.

Once an error has occurred and $TRAP has captured the contents of processor storage, you must IPL the system in order to continue processing.

The $DUMP utility formats and prints the data saved by $TRAP.

Considerations When Using $TRAP

Consider the following when you use $TRAP:

- You must load $TRAP in partition 1.
- $TRAP must be active when an error occurs. For this reason, you should use $TRAP when a problem exists and you are confident that you can reproduce the problem. For example, all programs that were active when the problem occurred should be active and in the same partitions. In addition, all devices that were switched on at the time the problem occurred should also be switched on when you start $TRAP.
- $TRAP dumps all of physical storage residing on your system. For this reason, make sure your dump data set is large enough to hold all mapped and unmapped storage.

If you cannot use $TRAP, you should perform a stand-alone dump of processor storage and send the diskette(s) with an APAR to IBM. The *Operation Guide* explains the procedure for taking a stand-alone dump. If you are preparing an APAR and you wish to send IBM a copy of your $TRAP or stand alone dump data set, you may need to send two or more diskettes. If your $TRAP data set was dumped to disk, you will need to follow the procedure in the *Operation Guide* on "Copying a $TRAP Data Set to Two Diskettes."

$TRAP - Save Storage on Error Condition *(continued)*

Allocating a Data Set

$TRAP dumps the mapped and unmapped storage areas of the system to a work data set which you supply. You have three options available for creating the data set:

1. Disk — You must create the disk data set at least as large as your system's physical storage.

2. One Diskette — You must create the diskette data set at least as large as your system's physical storage. You can use one diskette if your storage is not greater than 512K.

3. Two Diskettes — If your storage is greater than 512K, you must create two identical diskettes by using primary option 0 of the $DASDI utility. The name and volume must be the same for both diskettes ($$EDXLIB,IBMEDX).

   Note: You may want to number the diskettes "1" and "2" so you can distinguish them more easily later on.

When dumping to diskettes created by using primary option 0 of the $DASDI utility, it is not necessary to allocate the $TRAP work data set. However, if you are not dumping to these diskettes, you must calculate the data set size needed and allocate the data set using the $DISKUT1 utility.

To determine the size of the work data set, multiply the amount of physical storage in your system by four. For example, if you have 512K bytes of physical storage, you must allocate a data set of at least 2048 records. With 1024K bytes of physical storage, the $TRAP work data set must be a minimum size of 4096 records.

Loading $TRAP

You must load $TRAP in partition 1. To load the utility, press the attention key and type $L $TRAP. As shown in the following example, the system requests the name of the $TRAP work data set and volume.

```
> $L $TRAP
 DUMP  (NAME,VOLUME): TRAPWORK,MYVOL
LOADING $TRAP 22P, LP= BA00, PART=1
```

Note:

If your dump data set resides on diskette and you are using a 4966, the diskette must be in slot one (1).

$TRAP - Save Storage on Error Condition *(continued)*

Selecting The Conditions for the Dump

After you load $TRAP and enter the name of the work data set, the utility displays the $TRAP commands. Before entering a command, such as TRAPON, you must specify under what conditions you want the $TRAP dump to be taken. The utility displays the conditions, one at a time, and asks if it should dump processor storage on each condition. You can respond Y (YES) to one or all of the conditions shown here:

Machine Check: Indicates that your system has a hardware problem. If you respond Y to this prompt, the utility saves the hardware registers and storage when a storage parity, CPU control, or I/O check occurs. For an explanation of these errors, refer to the description manual for your processor.

Program Check: Indicates an error in an application program. If you respond Y to this prompt, the utility prompts you for the type of program check to trap. You can select any combination of the following five types:

- **Specification Check** — Occurs if a storage address violates boundary requirements. For example, a specification check occurs if a program attempts to move a word of data to an odd-byte boundary.

- **Invalid Storage Address** — Occurs when a program attempts to refer to a storage address outside of the storage size of the partition. This error occurs, for example, if a program attempts to do a cross-partition move to a nonexistent partition.

- **Privilege Violate** — Occurs if a program in problem state attempts to issue a privileged instruction. A privileged instruction can execute only if the program is in supervisor state.

  Normally, this error never occurs in an EDL program.

- **Protect Check** — Occurs if a program attempts to use protected storage. The processor can control access to areas in storage by using a storage protect feature.

  Normally, this error would never occur in an EDL program.

- **Invalid Function (Program Check)** — Occurs if the system attempts to execute an illegal operation code.

$TRAP - Save Storage on Error Condition *(continued)*

Soft Exception: Indicates a software error that the system software is equipped to handle. If you respond Y to this prompt, the system prompts you for the type of soft exception to trap. You can select any combination of the following three types:

- **Invalid Function (Soft Exception)** — Occurs if a program attempts to execute an instruction associated with a feature that is not contained in the supervisor. This error can occur, for example, if an EDL program attempts to use floating-point instructions (FADD, FSUB, FMULT, or FDIVD) when the supervisor does not support floating-point.

- **Floating-Point Exception** — Occurs when the optional floating-point processor detects an error condition. An EDL program can detect such an error condition by testing the return code from a floating-point instruction.

  Respond N to this prompt if your system does not have floating-point hardware.

- **Stack Exception** — Occurs when a program attempts to move an operand from an empty processor storage stack into a register or to move an operand from a register into a full processor storage stack. A stack exception also occurs when the stack cannot contain the number of words to be stored by an assembler Store Multiple (STM) instruction.

  Normally, this error never occurs in an EDL program.

Console Interrupt: Respond Y to this prompt if you want to be able to use the console interrupt button on the programmer console to force a $TRAP dump.

Save Floating-Point Registers: Respond Y to this prompt to save the contents of the floating-point registers when an error condition occurs. Respond N to this prompt if your system does not have floating-point hardware.

$TRAP - Save Storage on Error Condition *(continued)*

Starting, Suspending, and Ending $TRAP

Once you've selected the error conditions to trap, the utility displays the following messages:

```
$TRAP IS NOW SUSPENDED.  PRESS THE ATTENTION KEY (>) AND
 ENTER ONE OF THE FOLLOWING COMMANDS:
            -TO START $TRAP:    TRAPON
            -TO FORCE A DUMP:   TRAPDUMP
            -TO SUSPEND $TRAP:  TRAPOFF
            -TO END $TRAP:      TRAPEND
```

You can "start" $TRAP now by pressing the attention key and entering TRAPON. The system returns the message: "$TRAP STARTED."

The TRAPON command activates the trap utility but does not produce a storage dump. After starting $TRAP in partition 1, you should reload all programs and start all devices that were running when the error occurred. Remember to load the programs into the same partitions that they were in when you noticed the problem.

If $TRAP encounters the error you specify, it dumps the hardware registers and the contents of processor storage to the data set you specified when you loaded $TRAP. The dump is complete when your console lights display 'FFFF' Refer to "Dumping to Multiple Diskettes" on page UT-597 for an explanation of how to dump to two diskettes. You must then IPL the system to continue processing. Note the load points of the programs so you can display program storage if necessary with $DUMP. Use $DUMP to retrieve, format, and print the contents of the $TRAP work data set.

If you have a programmer console and you run your Series/1 in Diagnostic mode, you will receive a stop code of E9 if the $TRAP dump is successful.

Note:

If you set $TRAP to trap a program check, the program check message will not appear when you take the $TRAP dump. In addition, if a program has a task error exit routine, that routine will not receive control.

To **suspend** the $TRAP utility at any point, change to partition 1, press the attention key and enter TRAPOFF. The system returns the message: "$TRAP SUSPENDED (TRAPON RESTARTS $TRAP)." This command stops $TRAP until you start the utility again with the TRAPON command.

To **end** or cancel the $TRAP utility, press the attention key and enter the TRAPEND command in partition 1. The system returns the message: "$TRAP TERMINATED." Do **not** use the $C operator command to cancel $TRAP.

Forcing a $TRAP Dump

If an error condition (detectable by $TRAP) does not occur while $TRAP is active, you can still force a dump of storage by pressing the attention key and entering TRAPDUMP in partition 1. If

$TRAP - Save Storage on Error Condition *(continued)*

you have a programmer console, you can force a dump by pressing the console interrupt button. (To use this option, you must have replied Y to the prompt "DUMP ON CONSOLE INTERRUPT? (Y/N).")

Example 1

The following example shows the various options of the $TRAP utility. A description of each of the numbered items follows the example.

```
1    > $L $TRAP
2     DUMP  (NAME,VOLUME): TRAPWRK,CL001
      LOADING $TRAP 22P, LP= BA00, PART=1

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

     TO END $TRAP BEFORE A DUMP IS TAKEN, PRESS THE ATTENTION KEY (>)
      AND ENTER:  TRAPEND

          -- DO NOT USE $C TO END THIS UTILITY --

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

     SELECT ONE OR MORE OF THE FOLLOWING CONDITIONS:

3    DUMP ON MACHINE CHECK (Y/N)?  Y
3    DUMP ON PROGRAM CHECK (Y/N)?  Y
             SPECIFICATION CHECK (Y/N)?  Y
             INVALID STORAGE ADDRESS (Y/N)?  Y
             PRIVILEGE VIOLATE (Y/N)?  Y
             PROTECT CHECK (Y/N)?  Y
             INVALID FUNCTION (Y/N)?  Y
3    DUMP ON SOFT EXCEPTION (Y/N)?  Y
             INVALID FUNCTION (Y/N)?  Y
             FLOATING-POINT EXCEPTION (Y/N)?  Y
             STACK EXCEPTION (Y/N)?  Y
4    DUMP ON CONSOLE INTERRUPT (Y/N)?  Y

5    SAVE FLOATING-POINT REGISTERS (Y/N)?  Y

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

6    $TRAP IS NOW SUSPENDED.  PRESS THE ATTENTION KEY (>) AND
      ENTER ONE OF THE FOLLOWING COMMANDS:
                 -TO START $TRAP:    TRAPON
                 -TO FORCE A DUMP:   TRAPDUMP
                 -TO SUSPEND $TRAP:  TRAPOFF
                 -TO END $TRAP:      TRAPEND

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

     *****   NOTE:  IF A DUMP OCCURS, YOU MUST IPL THE SYSTEM   *****

7    > TRAPON
     $TRAP STARTED
```

$TRAP - Save Storage on Error Condition *(continued)*

1 Load $TRAP.

2 Enter the data set name and volume of the $TRAP work data set.

3 Set $TRAP to dump storage when it encounters a machine check, program check, or soft exception.

4 Indicate that you want the option of using the console interrupt button to force a $TRAP dump.

5 Indicate that you want to save the floating-point registers (if you have floating-point hardware installed).

6 The utility initially sets itself to "off."

7 Start $TRAP. Duplicate the problem you wish to trap.

In the following example, $TRAP is set to dump the hardware registers and the contents of processor storage when a specification check occurs. When the specification check does not occur, the operator suspends the utility and then ends it. A description of each of the numbered items follows the example.

$TRAP - Save Storage on Error Condition *(continued)*

Example 2

```
1     > $L $TRAP
       DUMP  (NAME,VOLUME): TRAPIT,EDX003
2      LOADING $TRAP 22P, LP= BA00, PART=1

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

      TO END $TRAP BEFORE A DUMP IS TAKEN, PRESS THE ATTENTION KEY (>)
       AND ENTER:  TRAPEND

           -- DO NOT USE $C TO END THIS UTILITY --

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

      SELECT ONE OR MORE OF THE FOLLOWING CONDITIONS:

      DUMP ON MACHINE CHECK (Y/N)?  N
3     DUMP ON PROGRAM CHECK (Y/N)?  Y
3             SPECIFICATION CHECK (Y/N)?  Y
              INVALID STORAGE ADDRESS (Y/N)?  N
              PRIVILEGE VIOLATE (Y/N)?  N
              PROTECT CHECK (Y/N)?  N
              INVALID FUNCTION (Y/N)?  N
      DUMP ON SOFT EXCEPTION (Y/N)?  N
      DUMP ON CONSOLE INTERRUPT (Y/N)?  N

      SAVE FLOATING-POINT REGISTERS (Y/N)?  N

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

      $TRAP IS NOW SUSPENDED.  PRESS THE ATTENTION KEY (>) AND
       ENTER ONE OF THE FOLLOWING COMMANDS:
                  -TO START $TRAP:    TRAPON
                  -TO FORCE A DUMP:   TRAPDUMP
                  -TO SUSPEND $TRAP:  TRAPOFF
                  -TO END $TRAP:      TRAPEND

      *****   NOTE:  IF A DUMP OCCURS, YOU MUST IPL THE SYSTEM   *****

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

4     > TRAPON
      $TRAP STARTED
5     > TRAPOFF
      $TRAP SUSPENDED (TRAPON RESTARTS $TRAP)
6     > TRAPEND
      $TRAP TERMINATED
```

$TRAP - Save Storage on Error Condition *(continued)*

1 Load $TRAP.

2 Enter the data set name and volume of the $TRAP work data set.

3 Set $TRAP to dump storage when it encounters a specification check.

4 Start $TRAP.

5 Suspend $TRAP. If the error occurs before the operator enters this command, the operator must IPL the system.

6 End $TRAP. The $DUMP utility formats and prints the data saved by $TRAP.

Dumping to Multiple Diskettes

To dump to multiple diskettes, mount the first $DASDI, primary-option-0-formatted diskette, load $TRAP, and reply to the prompts as in example 2 with the exception of DUMP (NAME,VOLUME). The NAME,VOLUME will be $$EDXLIB,IBMEDX. Then when a specification check occurs, $TRAP starts dumping to $$EDXLIB,IBMEDX. The programmer console lights indicate one of the following:

FFF2	Insert the next diskette in the same slot where you mounted the first one. The dump completes automatically.
FFFF	The dump is complete; IPL the system.
1111	Invalid format on the diskette. There is a possibility that you have used the wrong diskette.

Use the $DUMP utility to format and print the data saved by $TRAP.

$UPDATE - Converting Series/1 Programs

The $UPDATE conversion utility converts an object module into an executable, relocatable load module.

The object module used as input to $UPDATE may have been compiled by the Event Driven Language compiler $EDXASM or the host assembler. Object modules created by the host assembler must be transmitted to a Series/1 disk or diskette volume by a facility such as the IBM 2780/3780 RJE emulation utility $RJE2780/$RJE3780, or by the $HCFUT1 utility, before you can use them as the input to $UPDATE.

The object output of language translators other than $EDXASM or the host assembler must be processed by the linkage editor, $EDXLINK

Required Data Sets

$UPDATE requires the following data sets. You must specify these data sets when you invoke $UPDATE.

1. The *object-input data set* is the output of $EDXASM and serves as input to $UPDATE.

2. The *program-output data set* is the data set where the output of $UPDATE is to be placed. $UPDATE allocates this data set if it does not exist.

Invoking $UPDATE

You invoke $UPDATE with the $L operator command, option 2.9 of the session manager, or with the batch job stream processor ($JOBUTIL).

Updating a Program Using the $L Operator Command

When you invoke $UPDATE with the $L operator command, it prompts you for the information it requires. Examples of this interactive usage follow.

Invoking $UPDATE Using the $L Operator Command

This example shows $UPDATE being loaded for execution and the subsequent prompt messages.

```
> $L $UPDATE
LOADING $UPDATE     35P,00:17:51, LP= 4800, PART=1

THE DEFINED INPUT VOLUME IS EDX002, OK?  Y
THE DEFINED OUTPUT VOLUME IS EDX002, OK?  Y

COMMAND (?):
```

$UPDATE - Converting Series/1 Programs *(continued)*

$UPDATE Commands

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

```
COMMAND (?):  ?

CV - CHANGE VOLUME
RP - READ PROGRAM
EN - END

COMMAND (?):
```

Each command and its explanation is presented in alphabetical order on the following pages.

CV — Change Volume

Use the CV command to change the input or output volume. For example:

```
COMMAND (?):  CV

THE DEFINED INPUT VOLUME IS EDX002, OK?  N
ENTER NEW INPUT VOLUME LABEL:  EDX003
THE DEFINED OUTPUT VOLUME IS EDX003, OK?  Y

COMMAND (?):
```

If the volume is not mounted:

```
COMMAND (?):  CV

THE DEFINED INPUT VOLUME IS EDX003, OK?  N
ENTER NEW INPUT VOLUME LABEL:  EDXDFG
VOLUME OFF LINE

THE DEFINED INPUT VOLUME IS EDX003, OK?
```

$UPDATE - Converting Series/1 Programs *(continued)*

EN — End $UPDATE

Use the EN command to end $UPDATE processing.

Example:

```
COMMAND (?):  EN

$UPDATE ENDED AT 11:39:34
```

RP — Read and Store a Program

Use the RP command to read (convert) and store a program in an allocated data set. If the data set targeted for receiving output from the conversion process is not allocated, $UPDATE allocates the data set.

Example 1: Convert and store a new program. This example allocates a new data set TESTPROG (type PGM) of the required size if a data set with that name has not already been allocated.

```
COMMAND (?):  RP

OBJECT MODULE NAME:  OBJSET

OUTPUT PGM NAME:  TESTPROG

TESTPROG STORED

COMMAND (?):
```

If you specify the name of an input object module that *does not* exist, $UPDATE takes no action. For example:

```
COMMAND (?):  RP

OBJECT MODULE NAME:  DUMMY

OUTPUT PGM NAME:  PROGRAM

 INPUT DATA SET 'DUMMY' NOT FOUND
COMMAND (?):
```

In addition, an appropriate error message is issued if the output data set was not defined as a program type.

$UPDATE - Converting Series/1 Programs *(continued)*

Example 2: Convert and store a program where program and object module names are to be the same. In this example, an asterisk (*) is entered in place of the output data set name if the name is to be the same as the input data set name.

```
COMMAND (?): RP

OBJECT MODULE NAME: OBJSET

OUTPUT PGM NAME: *

PROGRAM NAME DEFAULTED TO OBJSET

OBJSET STORED ON EDX003

COMMAND (?):
```

Example 3: Convert and replace existing output program with same output name. In this example, an existing output program is replaced with a new output program. If the new and old sizes are the same, the new program data replaces the old with no other changes. If the new space required is different from the existing space, the existing data set is deleted and a new one of the proper size is allocated wherever enough free space is available.

```
COMMAND (?): RP

OBJECT MODULE NAME: OBJSET

OUTPUT PGM NAME: TSTPRG1

TSTPRG1 REPLACE? Y

TSTPRG1 STORED ON EDX002

COMMAND (?):
```

Example 4: Convert and rename new output program if an output program already exists. The existing output data set is undisturbed and a new data set (type PGM) of the proper size and with the new name is allocated.

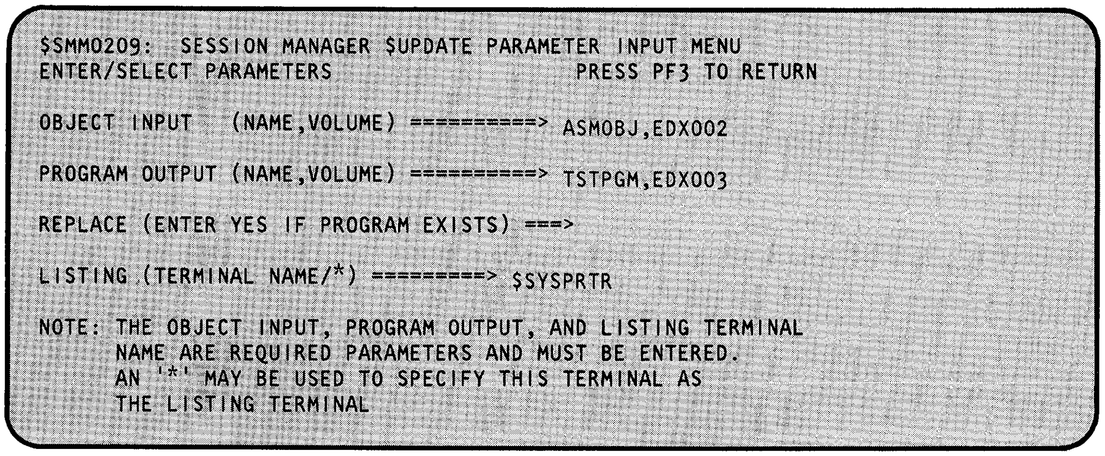
```
COMMAND (?): RP

OBJECT MODULE NAME: OBJSET

OUTPUT PGM NAME: TESTPROG
TESTPROG REPLACE? N
RENAME? Y

NEW PGM NAME: TSTPRG1

TSTPRG1 STORED ON EDX003

COMMAND (?):
```

$UPDATE - Converting Series/1 Programs *(continued)*

Updating a Program Using the Session Manager

To invoke $UPDATE using the session manager, select option 2.9.

Figure 39 shows the parameter input menu for entry of the required data sets and other parameters. The object program in data set ASMOBJ is to be formatted and placed in data set TSTPGM. The REPLACE parameter is left blank when a member does not already exist. If the member does exist, enter YES.



```
$SMM0209:  SESSION MANAGER $UPDATE PARAMETER INPUT MENU
ENTER/SELECT PARAMETERS                    PRESS PF3 TO RETURN

OBJECT INPUT   (NAME,VOLUME) ==========> ASMOBJ,EDX002

PROGRAM OUTPUT (NAME,VOLUME) ==========> TSTPGM,EDX003

REPLACE (ENTER YES IF PROGRAM EXISTS) ===>

LISTING (TERMINAL NAME/*) =========> $SYSPRTR

NOTE: THE OBJECT INPUT, PROGRAM OUTPUT, AND LISTING TERMINAL
      NAME ARE REQUIRED PARAMETERS AND MUST BE ENTERED.
      AN '*' MAY BE USED TO SPECIFY THIS TERMINAL AS
      THE LISTING TERMINAL
```

Figure 39. $UPDATE parameter input menu

Updating a Program Using $JOBUTIL

When you use the job-stream processor ($JOBUTIL) to invoke $UPDATE, the information that $UPDATE requires must be passed by the PARM command of $JOBUTIL. The required information consists of:

1. The name of the device to receive the printed output resulting from $UPDATE execution.

2. The name,volume of the data set containing the input object module. The volume name, if omitted, defaults to the IPL volume.

3. The name,volume of the data set to contain the output loadable program. The volume name, if omitted, defaults to the IPL volume.

4. The optional parameter YES if the output module is to replace an existing module of the same name,volume.

The first three items of information are required and must be given in the order described. At least one blank must occur between each of these four items in the PARM command.

When you invoke $UPDATE under $JOBUTIL, the system assumes you mean the "read program" (RP) command. The "rename" function is not supported, but the "replace" function is. Refer to the preceding examples for a description of rename and replace.

$UPDATE - Converting Series/1 Programs *(continued)*

In batch mode, $UPDATE terminates after performing one RP command. A completion code is set by $UPDATE depending upon the success or failure of the requested operation. You can test this code by the JUMP command of $JOBUTIL.

Note: The $UPDATE completion codes are described in the *Messages and Codes* book.

The following is an example of invoking $UPDATE via $JOBUTIL commands.

```
 .
 .
 .
PROGRAM    $UPDATE
PARM       $SYSPRTR  OBJMOD,VOL1  MYPROG  YES
NOMSG
EXEC
 .
 .
 .
```

In this example, $SYSPRTR receives the printed messages, the input object module is OBJMOD on VOL1, and the output program is MYPROG on the IPL volume. If MYPROG already exists on the IPL volume, it is replaced by the new version. If MYPROG does not already exist, $UPDATE allocates space for it.

$UPDATE Output

The output load module is stored in the output data set upon successful completion of the RP command of $UPDATE.

Considerations When Creating a Supervisor

You can create multiple supervisor programs for different machine configurations on one Series/1. You can then copy them to diskettes which can be used on the Series/1 having the proper configuration.

$UPDATE - Converting Series/1 Programs *(continued)*

The name $EDXNUC for the output program receives special treatment by $UPDATE since the creation of a supervisor results in an absolute rather than a relocatable program. The following rules apply:

1. If the first seven characters of the output program name are $EDXNUC, then an absolute supervisor program is formatted.

2. The eighth character can be a blank or any other character. The output program replaces an existing program or creates a new program by that name on the specified volume.

3. An existing supervisor program is not deleted and reallocated by the update program. The new version must fit in the existing space.

To test supervisors created and stored by $UPDATE, either:

- Copy the new supervisor nucleus into the IPL supervisor on the IPL volume and IPL the system again, or

- Rewrite the IPL text on the volume on which the supervisor is located using the $INITDSK utility option II.

$UPDATEH - Converting Host System Programs

$UPDATEH transfers, over a communications link, Series/1 object programs that are members of a host-partitioned data set (PDS) and stores them in a Series/1 disk or diskette volume in the proper format to be loaded for execution. The programs must have been assembled previously on the host.

To change the name of the default host library, locate the label "HOSTNAME" in the $UPDATEH listing and change the name to the host library name you want. Then you must assemble $UPDATEH and install it in the program library.

$UPDATEH requires that the Event Driven Executive Host Communication Facility (IUP 5796-PGH) be installed on the host computer.

Invoking $UPDATEH

You invoke $UPDATEH with the $L operator command, option 2.10 of the session manager, or with the batch job-stream processor ($JOBUTIL).

Updating a Hosting Program Using the $L Operator Command

When you invoke $UPDATEH with the $L operator command, it prompts you for the information it requires. Examples of this interactive usage follow.

This example shows $UPDATEH being loaded with the $L operator command and the subsequent messages that follow.

```
> $L
PGM(NAME,VOLUME):  $UPDATEH
LOADING $UPDATEH     28P,15:29:59, LP=4400, PART=2

THE DEFINED HOST LIBRARY IS S1.EDX.LOADLIB OK (Y/N)?  Y
THE DEFINED VOLUME IS EDX002,OK (Y/N)?  Y

COMMAND (?):
```

$UPDATEH Commands

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

```
COMMAND (?):  ?

CH     CHANGE HOST LIBRARY
CV     CHANGE VOLUME
RP     READ PROGRAM
EN     END

COMMAND (?):
```

$UPDATEH - Converting Host System Programs *(continued)*

After $UPDATEH displays the commands, it prompts you with COMMAND (?): again. Then you can respond with the command of your choice (for example, CH). Each command and its explanation is presented in alphabetical order on the following pages.

CH — Change Host Library

Use the CH command to change the host library. The system prompts you for the new host library name.

Example: Change the host library.

```
COMMAND (?):  CH

ENTER HOST LIBRARY NAME:  S1.EDX.LOADLIB2

COMMAND (?):
```

CV — Change Volume

Use the CV command to change the volume. The system prompts you for the new volume label.

Example: Change the volume.

```
COMMAND (?):  CV

ENTER VOLUME LABEL:  EDX003

COMMAND (?):
```

EN — End $UPDATEH

Use the EN command to end the $UPDATEH utility.

Example: End $UPDATEH.

```
COMMAND (?):  EN

$UPDATEH ENDED 15.30.54
```

$UPDATEH - Converting Host System Programs *(continued)*

RP — Read a Program

Use the RP command to transfer a new program to a Series/1 data set, transfer a program and rename it, or transfer and replace an existing program.

Example 1: Transfer a new program to a Series/1 data set.

```
COMMAND (?):  RP

PROGRAM NAME:  TPTEST
PROGRAM: TPTEST STORED

COMMAND(?):
```

Example 2: Transfer and replace an existing program.

```
COMMAND (?):  RP

PROGRAM NAME:  PROG1
PROG1 REPLACE (Y/N)?  Y
PROGRAM: PROG1 STORED

COMMAND(?):
```

Example 3: Transfer a program PROG1 and rename it PROG2. The existing program (PROG1) is not replaced.

```
COMMAND (?):  RP

PROGRAM NAME:  PROG1
PROGRAM: PROG1 REPLACE (Y/N)?  N
RENAME (Y/N)?  Y

PROGRAM NAME:  PROG2
PROGRAM: PROG2 STORED

COMMAND(?):
```

Updating a Host Program Using $JOBUTIL or Session Manager

When you load $UPDATEH with the job-stream processor $JOBUTIL or the session manager, it prompts you for control information. Execution is controlled as described above.

Notes

Glossary of Terms and Abbreviations

This glossary defines terms and abbreviations used in the Series/1 Event Driven Executive software publications. All software and hardware terms pertain to EDX. This glossary also serves as a supplement to the *IBM Data Processing Glossary*, GC20-1699.

$SYSLOGA, $SYSLOGB. The name of the alternate system logging device. This device is optional but, if defined, should be a terminal with keyboard capability, not just a printer.

$SYSLOG. The name of the system logging device or operator station; must be defined for every system. It should be a terminal with keyboard capability, not just a printer.

$SYSPRTR. The name of the system printer.

abend. Abnormal end-of-task. Termination of a task prior to its completion because of an error condition that cannot be resolved by recovery facilities while the task is executing.

ACCA. See asynchronous communications control adapter.

address key. Identifies a set of Series/1 segmentation registers and represents an address space. It is one less than the partition number.

address space. The logical storage identified by an address key. An address space is the storage for a partition.

application program manager. The component of the Multiple Terminal Manager that provides the program management facilities required to process user requests. It controls the contents of a program area and the execution of programs within the area.

application program stub. A collection of subroutines that are appended to a program by the linkage editor to provide the link from the application program to the Multiple Terminal Manager facilities.

asynchronous communications control adapter. An ASCII terminal attached via #1610, #2091 with #2092, or #2095 with #2096 adapters.

attention key. The key on the display terminal keyboard that, if pressed, tells the operating system that you are entering a command.

attention list. A series of pairs of 1 to 8 byte EBCDIC strings and addresses pointing to EDL instructions. When the attention key is pressed on the terminal, the operator can enter one of the strings to cause the associated EDL instructions to be executed.

backup. A copy of data to be used in the event the original data is lost or damaged.

base record slots. Space in an indexed file that is reserved for based records to be placed.

base records. Records are placed into an indexed file while in load mode or inserted in process mode with a new high key.

basic exchange format. A standard format for exchanging data on diskettes between systems or devices.

binary synchronous device data block (BSCDDB). A control block that provides the information to control one Series/1 Binary Synchronous Adapter. It determines the line characteristics and provides dedicated storage for that line.

Glossary of Terms and Abbreviations

block. (1) See data block or index block. (2) In the Indexed Method, the unit of space used by the access method to contain indexes and data.

block mode. The transmission mode in which the 3101 Display Station transmits a data data stream, which has been edited and stored, when the SEND key is pressed.

BSCAM. See binary synchronous communications access method.

binary synchronous communications access method. A form of binary synchronous I/O control used by the Series/1 to perform data communications between local or remote stations.

BSCDDB. See binary synchronous device data block.

buffer. An area of storage that is temporarily reserved for use in performing an input/output operation, into which data is read or from which data is written. See input buffer and output buffer.

bypass label processing. Access of a tape without any label processing support.

CCB. See terminal control block.

central buffer. The buffer used by the Indexed Access Method for all transfers of information between main storage and indexed files.

character image. An alphabetic, numeric, or special character defined for an IBM 4978 Display Station. Each character image is defined by a dot matrix that is coded into eight bytes.

character image table. An area containing the 256 character images that can be defined for an IBM 4978 Display Station. Each character image is coded into eight bytes, the entire table of codes requiring 2048 bytes of storage.

character mode. The transmission mode in which the 3101 Display Station immediately sends a character when a keyboard key is pressed.

cluster. In an indexed file, a group of data blocks that is pointed to from the same primary-level index block, and includes the primary-level index block. The data records and blocks contained in a cluster are logically contiguous, but are not necessarily physically contiguous.

COD (change of direction). A character used with ACCA terminal to indicate a reverse in the direction of data movement.

cold start. Starting the spool facility by erasing any spooled jobs remaining in the spool data set from any previous spool session.

command. A character string from a source external to the system that represents a request for action by the system.

common area. A user-defined data area that is mapped into the partitions specified on the SYSTEM definition statement. It can be used to contain control blocks or data that will be accessed by more than one program.

completion code. An indicator that reflects the status of the execution of a program. The completion code is displayed or printed on the program's output device.

constant. A value or address that remains unchanged thoughout program execution.

controller. A device that has the capability of configuring the GPIB bus by designating which devices are active, which devices are listeners, and which device is the talker. In Series/1 GPIB implementation, the Series/1 is always the controller.

conversion. See update.

control station. In BSCAM communications, the station that supervises a multipoint connection, and performs polling and selection of its tributary stations. The status of control station is assigned to a BSC line during system generation.

cross-partition service. A function that accesses data in two partitions.

cross-partition supervisor. A supervisor in which one or more supervisor modules reside outside of partition 1 (address space 0).

data block. In an indexed file, an area that contains control information and data records. These blocks are a multiple of 256 bytes.

data record. In an indexed file, the records containing customer data.

data set. A group of records within a volume pointed to by a directory member entry in the directory for the volume.

data set control block (DSCB). A control block that provides the information required to access a data set, volume or directory using READ and WRITE.

data set shut down. An indexed data set that has been marked (in main storage only) as unusable due to an error.

DCE. See directory control entry.

device data block (DDB). A control block that describes a disk or diskette volume.

direct access. (1) The access method used to READ or WRITE records on a disk or diskette device by specifying their location relative the beginning of the data set or volume. (2) In the Indexed Access Method, locating any record via its key without respect to the previous operation. (3) A condition in terminal I/O where a READTEXT or a PRINTEXT is directed to a buffer which was previously enqueued upon by an IOCB.

directory. (1) A series of contiguous records in a volume that describe the contents in terms of allocated data sets and free space. (2) A series of contiguous records on a device that describe the contents in terms of allocated volumes and free space. (3) For the Indexed Access Method Version 2, a data set that defines the relationship between primary and secondary indexed files (secondary index support).

directory control entry (DCE). The first 32 bytes of the first record of a directory in which a description of the directory is stored.

directory member entry (DME). A 32-byte directory entry describing an allocated data set or volume.

display station. An IBM 4978, 4979, or 3101 display terminal or similar terminal with a keyboard and a video display.

DME. See directory member entry.

DSCB. See data set control block.

dynamic storage. An increment of storage that is appended to a program when it is loaded.

end-of-data indicator. A code that signals that the last record of a data set has been read or written. End-of-data is determined by an end-of-data pointer in the DME or by the physical end of the data set.

ECB. See event control block.

EDL. See Event Driven Language.

emulator. The portion of the Event Driven Executive supervisor that interprets EDL instructions and performs the function specified by each EDL statement.

end-of-tape (EOT). A reflective marker placed near the end of a tape and sensed during output. The marker signals that the tape is nearly full.

enter key. The key on the display terminal keyboard that, if pressed, tells the operating system to read the information you entered.

event control block (ECB). A control block used to record the status (occurred or not occurred) of an event; often used to synchronize the execution of tasks. ECBs are used in conjunction with the WAIT and POST instructions.

Event Driven Language (EDL). The language for input to the Event Driven Executive compiler ($EDXASM), or the Macro and Host assemblers in conjunction with the Event Driven Executive macro libraries. The output is interpreted by the Event Driven Executive emulator.

EXIO (execute input or output). An EDL facility that provides user controlled access to Series/1 input/output devices.

external label. A label attached to the outside of a tape that identifies the tape visually. It usually contains items of identification such as file name and number, creation data, number of volumes, department number, and so on.

external name (EXTRN). The 1- to 8-character symbolic EBCDIC name for an entry point or data field that is not defined within the module that references the name.

FCA. See file control area.

FCB. See file control block.

file. A set of related records treated as a logical unit. Although file is often used interchangeably with data set, it usually refers to an indexed or a sequential data set.

file control area (FCA). A Multiple Terminal Manager data area that describes a file access request.

file control block (FCB). The first block of an indexed file. It contains descriptive information about the data contained in the file.

file control block extension. The second block of an indexed file. It contains the file definition parameters used to define the file.

file manager. A collection of subroutines contained within the program manager of the Multiple Terminal Manager that provides common support for all disk data transfer operations as needed for transaction-oriented application programs. It supports indexed and direct files under the control of a single callable function.

floating point. A positive or negative number that can have a decimal point.

formatted screen image. A collection of display elements or display groups (such as operator prompts and field input names and areas) that are presented together at one time on a display device.

free pool. In an indexed data set, a group of blocks that can be used for either data blocks or index blocks. These differ from other free blocks in that these are not initially assigned to specific logical positions in the file.

free space. In an indexed file, records blocks that do not currently contain data, and are available for use.

free space entry (FSE). An 8-byte directory entry defining an area of free space within a volume or a device.

FSE. See free space entry.

general purpose interface bus. The IEEE Standard 488-1975 that allows various interconnected devices to be attached to the GPIB adapter (RPQ D02118).

Glossary of Terms and Abbreviations

GPIB. See general purpose interface bus.

group. A unit of 100 records in the spool data set allocated to a spool job.

H exchange format. A standard format for exchanging data on diskettes between systems or devices.

host assembler. The assembler licensed program that executes in a 370 (host) system and produces object output for the Series/1. The source input to the host assembler is coded in Event Driven Language or Series/1 assembler language. The host assembler refers to the System/370 Program Preparation Facility (5798-NNQ).

host system. Any system whose resources are used to perform services such as program preparation for a Series/1. It can be connected to a Series/1 by a communications link.

IACB. See indexed access control block.

IAR. See instruction address register.

ICB. See indexed access control block.

IIB. See interrupt information byte.

image store. The area in a 4978 that contains the character image table.

immediate data. A self-defining term used as the operand of an instruction. It consists of numbers, messages or values which are processed directly by the computer and which do not serve as addresses or pointers to other data in storage.

index. In an indexed file, an ordered collection of pairs of keys and pointers, used to sequence and locate records.

index block. In an indexed file, an area that contains control information and index entries. These blocks are a multiple of 256 bytes.

indexed access control block (IACB/ICB). The control block that relates an application program to an indexed file.

indexed access method. An access method for direct or sequential processing of fixed-length records by use of a record's key.

indexed data set. Synonym for indexed file.

indexed file. A file specifically created, formatted and used by the Indexed Access Method. An indexed file is sometimes called an indexed data set.

index entry. In an indexed file, a key-pointer pair, where the pointer is used to locate a lower-level index block or a data block.

index register (#1, #2). Two words defined in EDL and contained in the task control block for each task. They are used to contain data or for address computation.

input buffer. (1) See buffer. (2) In the Multiple Terminal Manager, an area for terminal input and output.

input output control block (IOCB). A control block containing information about a terminal such as the symbolic name, size and shape of screen, the size of the forms in a printer, or an optional reference to a user provided buffer.

instruction address register (IAR). The pointer that identifies the machine instruction currently being executed. The Series/1 maintains a hardware IAR to determine the Series/1 assembler instruction being executed. It is located in the level status block (LSB).

integer. A positive or negative number that has no decimal point.

interactive. The mode in which a program conducts a continuous dialogue between the user and the system.

internal label. An area on tape used to record identifying information (similar to the identifying information placed on an external label). Internal labels are checked by the system to ensure that the correct volume is mounted.

interrupt information byte (IIB). In the Multiple Terminal Manager, a word containing the status of a previous input/output request to or from a terminal.

invoke. To load and activate a program, utility, procedure, or subroutine into storage so it can run.

job. A collection of related program execution requests presented in the form of job control statements, identified to the jobstream processor by a JOB statement.

job control statement. A statement in a job that specifies requests for program execution, program parameters, data set definitions, sequence of execution, and, in general, describes the environment required to execute the program.

job stream processor. The job processing facility that reads job control statements and processes the requests made by these statements. The Event Driven Executive job stream processor is $JOBUTIL.

jumper. (1) A wire or pair of wires which are used for the arbitrary connection between two circuits or pins in an attachment card. (2) To connect wire(s) to an attachment card or to connect two circuits.

key. In the Indexed Access Method, one or more consecutive characters used to identify a record and establish its order with respect to other records. See also key field.

key field. A field, located in the same position in each record of an indexed file, whose content is used for the key of a record.

level status block (LSB). A Series/1 hardware data area that contains processor status. This area is eleven words in length.

library. A set of contiguous records within a volume. It contains a directory, data sets and/or available space.

line. A string of characters accepted by the system as a single input from a terminal; for example, all characters entered before the carriage return on the teletypewriter or the ENTER key on the display station is pressed.

link edit. The process of resolving external symbols in one or more object modules. A link edit is performed with $EDXLINK whose output is a loadable program.

listener. A controller or active device on a GPIB bus that is configured to accept information from the bus.

load mode. In the Indexed Access Method, the mode in which records are loaded into base record slots in an indexed file.

load module. A single module having cross references resolved and prepared for loading into storage for execution. The module is the output of the $UPDATE or $UPDATEH utility.

load point. (1) Address in the partition where a program is loaded. (2) A reflective marker placed near the beginning of a tape to indicate where the first record is written.

lock. In the Indexed Access Method, a method of indicating that a record or block is in use and is not available for another request.

logical screen. A screen defined by margin settings, such as the TOPM, BOTM, LEFTM and RIGHTM parameters of the TERMINAL or IOCB statement.

LSB. See level status block.

mapped storage. The processor storage that you defined on the SYSTEM statement during system generation.

member. A term used to identify a named portion of a partitioned data set (PDS). Sometimes member is also used as a synonym for a data set. See data set.

menu. A formatted screen image containing a list of options. The user selects an option to invoke a program.

menu-driven. The mode of processing in which input consists of the responses to prompting from an option menu.

message. In data communications, the data sent from one station to another in a single transmission. Stations communication with a series of exchanged messages.

multifile volume. A unit of recording media, such as tape reel or disk pack, that contains more than one data file.

multiple terminal manager. An Event Driven Executive licensed program that provides support for transaction-oriented applications on a Series/1. It provides the capability to define transactions and manage the programs that support those transactions. It also manages multiple terminals as needed to support these transactions.

multivolume file. A data file that, due to its size, requires more than one unit of recording media (such as tape reel or disk pack) to contain the entire file.

new high key. A key higher than any other key in an indexed file.

nonlabeled tapes. Tapes that do not contain identifying labels (as in standard labeled tapes) and contain only files separated by tapemarks.

null character. A user-defined character used to define the unprotected fields of a formatted screen.

option selection menu. A full screen display used by the Session Manager to point to other menus or system functions, one of which is to be selected by the operator. (See primary option menu and secondary option menu.)

output buffer. (1) See buffer. (2) In the Multiple Terminal Manager, an area used for screen output and to pass data to subsequent transaction programs.

overlay. The technique of reusing a single storage area allocated to a program during execution. The storage area can be reused by loading it with overlay programs that have been specified in the PROGRAM statement of the program or by calling overlay segments that have been specified in the OVERLAY statement of $EDXLINK.

overlay area. A storage area within a program reserved for overlay programs specified in the PROGRAM statement or overlay segments specified in the OVERLAY statement in $EDXLINK.

overlay program. A program in which certain control sections can use the same storage location at different times during execution. An overlay program can execute concurrently as an asynchronous task with other programs and is specified in the EDL PROGRAM statement in the main program.

overlay segment. A self-contained portion of a program that is called and sequentially executes as a synchronous task. The entire program that calls the overlay segment need not be maintained in storage while the overlay segment is executing. An overlay segment is specified in the OVERLAY statement of $EDXLINK or $XPSLINK (for initialization modules).

overlay segment area. A storage area within a program or supervisor reserved for overlay segments. An overlay segment area is specified with the OVLAREA statement of $EDXLINK.

Glossary of Terms and Abbreviations

parameter selection menu. A full screen display used by the Session Manager to indicate the parameters to be passed to a program.

partition. A contiguous fixed-sized area of storage. Each partition is a separate address space.

performance volume. A volume whose name is specified on the DISK definition statement so that its address is found during IPL, increasing system performance when a program accesses the volume.

physical timer. Synonym for timer (hardware).

polling. In data communications, the process by which a multipoint control station asks a tributary if it can receive messages.

precision. The number of words in storage needed to contain a value in an operation.

prefind. To locate the data sets or overlay programs to be used by a program and to store the necessary information so that the time required to load the prefound items is reduced.

primary file. An indexed file containing the data records and primary index.

primary file entry. For the Indexed Access Method Version 2, an entry in the directory describing a primary file.

primary index. The index portion of a primary file. This is used to access data records when the primary key is specified.

primary key. In an indexed file, the key used to uniquely identify a data record.

primary-level index block. In an indexed file, the lowest level index block. It contains the relative block numbers (RBNs) and high keys of several data blocks. See cluster.

primary menu. The program selection screen displayed by the Multiple Terminal Manager.

primary option menu. The first full screen display provided by the Session Manager.

primary station. In a Series/1-to-Series/1 Attachment, the processor that controls communication between the two computers. Contrast with secondary station.

primary task. The first task executed by the supervisor when a program is loaded into storage. It is identified by the PROGRAM statement.

priority. A combination of hardware interrupt level priority and a software ranking within a level. Both primary and secondary tasks will execute asynchronously within the system according to the priority assigned to them.

process mode. In the Indexed Access Method, the mode in which records can be retrieved, updated, inserted, or deleted.

processor status word (PSW). A 16-bit register used to (1) record error or exception conditions that may prevent further processing and (2) hold certain flags that aid in error recovery.

program. A disk- or diskette-resident collection of one or more tasks defined by a PROGRAM statement; the unit that is loaded into storage. (See primary task and secondary task.)

program header. The control block found at the beginning of a program that identifies the primary task, data sets, storage requirements and other resources required by a program.

program/storage manager. A component of the Multiple Terminal Manager that controls the execution and flow of application programs within a single program area and contains the support needed to allow multiple operations and sharing of the program area.

protected field. A field in which the operator cannot use the keyboard to enter, modify, or erase data.

PSW. See processor status word.

QCB. See queue control block.

QD. See queue descriptor.

QE. See queue element.

queue control block (QCB). A data area used to serialize access to resources that cannot be shared. See serially reusable resource.

queue descriptor (QD). A control block describing a queue built by the DEFINEQ instruction.

queue element (QE). An entry in the queue defined by the queue descriptor.

quiesce. To bring a device or a system to a halt by rejection of new requests for work.

quiesce protocol. A method of communication in one direction at a time. When sending node wants to receive, it releases the other node from its quiesced state.

record. (1) The smallest unit of direct access storage that can be accessed by an application program on a disk or diskette using READ and WRITE. Records are 256 bytes in length. (2) In the Indexed Access Method, the logical unit that is transferred between $IAM and the user's buffer. The length of the buffer is defined by the user. (3) In BSCAM communications, the portions of data transmitted in a message. Record length (and, therefore, message length) can be variable.

recovery. The use of backup data to re-create data that has been lost or damaged.

reflective marker. A small adhesive marker attached to the reverse (nonrecording) surface of a reel of magnetic tape. Normally, two reflective markers are used on each reel of tape. One indicates the beginning of the recording area on the tape (load point), and the other indicates the proximity to the end of the recording area (EOT) on the reel.

relative block address (RBA). The location of a block of data on a 4967 disk relative to the start of the device.

relative record number. An integer value identifying the position of a record in a data set relative to the beginning of the data set. The first record of a data set is record one, the second is record two, the third is record three.

relocation dictionary (RLD). The part of an object module or load module that is used to identify address and name constants that must be adjusted by the relocating loader.

remote management utility control block (RCB). A control block that provides information for the execution of remote management utility functions.

reorganize. The process of copying the data in an indexed file to another indexed file in a manner that rearranges the data for more optimum processing and free space distribution.

restart. Starting the spool facility w the spool data set contains jobs from a previous session. The jobs in the spool data set can be either deleted or printed when the spool facility is restarted.

return code. An indicator that reflects the results of the execution of an instruction or subroutine. The return code is usually placed in the task code word (at the beginning of the task control block).

roll screen. A display screen which is logically segmented into an optional history area and a work area. Output directed to the screen starts display at the beginning of the work area and continues on down in a line-by-line sequence. When the work area gets full, the operator presses ENTER/SEND and its contents are shifted into the optional history area and the work area itself is erased. Output now starts again at the beginning of the work area.

SBIOCB. See sensor based I/O control block.

second-level index block. In an indexed data set, the second-lowest level index block. It contains the addresses and high keys of several primary-level index blocks.

secondary file. See secondary index.

secondary index. For the Indexed Access Method Version 2, an indexed file used to access data records by their secondary keys. Sometimes called a secondary file.

secondary index entry. For the Indexed Access Method Version 2, this an an entry in the directory describing a secondary index.

secondary key. For the Indexed Access Method Version 2, the key used to uniquely identify a data record.

secondary option menu. In the Session Manager, the second in a series of predefined procedures grouped together in a hierarchical structure of menus. Secondary option menus provide a breakdown of the functions available under the session manager as specified on the primary option menu.

secondary task. Any task other than the primary task. A secondary task must be attached by a primary task or another secondary task.

secondary station. In a Series/1-to-Series/1 Attachment, the processor that is under the control of the primary station.

sector. The smallest addressable unit of storage on a disk or diskette. A sector on a 4962 or 4963 disk is equivalent to an Event Driven Executive record. On a 4964 or 4966 diskette, two sectors are equivalent to an Event Driven Executive record.

selection. In data communications, the process by which the multipoint control station asks a tributary station if it is ready to send messages.

self-defining term. A decimal, integer, or character that the computer treats as a decimal, integer, or character and not as an address or pointer to data in storage.

sensor based I/O control block (SBIOCB). A control block containing information related to sensor I/O operations.

sequential access. The processing of a data set in order of occurrence of the records in the data set. (1) In the Indexed Access Method, the processing of records in ascending collating sequence order of the keys. (2) When using READ/WRITE, the processing of records in ascending relative record number sequence.

serially reusable resource (SRR). A resource that can only be accessed by one task at a time. Serially reusable resources are usually managed via (1) a QCB and ENQ/DEQ statements or (2) an ECB and WAIT/POST statements.

service request. A device generated signal used to inform the GPIB controller that service is required by the issuing device.

session manager. A series of predefined procedures grouped together as a hierarchical structure of menus from which you select the utility functions, program preparation facilities, and language processors needed to prepare and execute application programs. The menus consist of a primary option menu that displays functional groupings and secondary option menus that display a breakdown of these functional groupings.

shared resource. A resource that can be used by more than one task at the same time.

Glossary of Terms and Abbreviations

shut down. See data set shut down.

source module/program. A collection of instructions and statements that constitute the input to a compiler or assembler. Statements may be created or modified using one of the text editing facilities.

spool job. The set of print records generated by a program (including any overlays) while engueued to a printer designated as a spool device.

spool session. An invocation and termination of the spool facility.

spooling. The reading of input data streams and the writing of output data streams on storage devices, concurrently with job execution, in a format convenient for later processing or output operations.

SRQ. See service request.

stand-alone dump. An image of processor storage written to a diskette.

stand-alone dump diskette. A diskette supplied by IBM or created by the $DASDI utility.

standard labels. Fixed length 80-character records on tape containing specific fields of information (a volume label identifying the tape volume, a header label preceding the data records, and a trailer label following the data records).

static screen. A display screen formatted with predetermined protected and unprotected areas. Areas defined as operator prompts or input field names are protected to prevent accidental overlay by input data. Areas defined as input areas are not protected and are usually filled in by an operator. The entire screen is treated as a page of information.

station. In BSCAM communications, a BSC line attached to the Series/1 and functioning in a point-to-point or multipoint connection. Also, any other terminal or processor with which the Series/1 communicates.

subroutine. A sequence of instructions that may be accessed from one or more points in a program.

supervisor. The component of the Event Driven Executive capable of controlling execution of both system and application programs.

system configuration. The process of defining devices and features attached to the Series/1.

SYSGEN. See system generation.

system generation. The processing of defining I/O devices and selecting software options to create a supervisor tailored to the needs of a specific Series/1 hardware configuration and application.

system partition. The partition that contains the root segment of the supervisor (partition number 1, address space 0).

talker. A controller or active device on a GPIB bus that is configured to be the source of information (the sender) on the bus.

tape device data block (TDB). A resident supervisor control block which describes a tape volume.

tapemark. A control character recorded on tape used to separate files.

task. The basic executable unit of work for the supervisor. Each task is assigned its own priority and processor time is allocated according to this priority. Tasks run independently of each other and compete for the system resources. The first task of a program is the primary task. All tasks attached by the primary task are secondary tasks.

task code word. The first two words (32 bits) of a task's TCB; used by the emulator to pass information from system to task regarding the outcome of various operations, such as event completion or arithmetic operations.

task control block (TCB). A control block that contains information for a task. The information consists of pointers, save areas, work areas, and indicators required by the supervisor for controlling execution of a task.

task supervisor. The portion of the Event Driven Executive that manages the dispatching and switching of tasks.

TCB. See task control block.

terminal. A physical device defined to the EDX system using the TERMINAL configuration statement. EDX terminals include directly attached IBM displays, printers and devices that communicate with the Series/1 in an asynchronous manner.

terminal control block (CCB). A control block that defines the device characteristics, provides temporary storage, and contains links to other system control blocks for a particular terminal.

terminal environment block (TEB). A control block that contains information on a terminal's attributes and the program manager operating under the Multiple Terminal Manager. It is used for processing requests between the terminal servers and the program manager.

terminal screen manager. The component of the Multiple Terminal Manager that controls the presentation of screens and communications between terminals and transaction programs.

terminal server. A group of programs that perform all the input/output and interrupt handling functions for terminal devices under control of the Multiple Terminal Manager.

terminal support. The support provided by EDX to manage and control terminals. See terminal.

timer. The timer features available with the Series/1 processors. Specifically, the 7840 Timer Feature card (4955 only) or the native timer (4952, 4954, and 4956). Only one or the other is supported by the Event Driven Executive.

trace range. A specified number of instruction addresses within which the flow of execution can be traced.

transaction oriented applications. Program execution driven by operator actions, such as responses to prompts from the system. Specifically, applications executed under control of the Multiple Terminal Manager.

transaction program. See transaction-oriented applications.

transaction selection menu. A Multiple Terminal Manager display screen (menu) offering the user a choice of functions, such as reading from a data file, displaying data on a terminal, or waiting for a response. Based upon the choice of option, the application program performs the requested processing operation.

tributary station. In BSCAM communications, the stations under the supervision of a control station in a multipoint connection. They respond to the control station's polling and selection.

unmapped storage. The processor storage in your processor that you did not define on the SYSTEM statement during system generation.

unprotected field. A field in which the operator can use the keyboard to enter, modify or erase data. Also called non-protected field.

update. (1) To alter the contents of storage or a data set. (2) To convert object modules, produced as the output of an assembly or compilation, or the output of the linkage editor, into a form that can be loaded into storage for program execution and to update the directory of the volume on which the loadable program is stored.

user exit. (1) Assembly language instructions included as part of an EDL program and invoked via the USER instruction. (2) A point in an IBM-supplied program where a user written routine can be given control.

variable. An area in storage, referred to by a label, that can contain any value during program execution.

vary offline. (1) To change the status of a device from online to offline. When a device is offline, no data set can be accessed on that device. (2) To place a disk or diskette in a state where it is unknown by the system.

vary online. To place a device in a state where it is available for use by the system.

vector. An ordered set or string of numbers.

volume. A disk, diskette, or tape subdivision defined using $INITDSK or $TAPEUT1.

volume descriptor entry (VDE). A resident supervisor control block that describes a volume on a disk or diskette.

volume label. A label that uniquely identifies a single unit of storage media.

Index

The following index contains entries for this book only. See the *Library Guide and Common Index* for a Common Index to all Event Driven Executive books.

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

IBM Series/1 Event Driven Executive

Publications Order Form

Instructions:

1. Complete the order form, supplying all of the requested information. (Please print or type.)
2. If you are placing the order by phone, dial **1-800-IBM-2468.**
3. If you are mailing your order, fold the order form as indicated, seal with tape, and mail. We pay the postage.

Ship to:

Name: ______________________

____

Address: ______________________

____

City: ______________________

State: ______________ Zip: ______________

Bill to:

Customer number: ______________________

Name: ______________________

____

Address: ______________________

____

City: ______________________

State: ______________ Zip: ______________

Your Purchase Order No.: ______________________

Phone: (   ) ______________________

Signature: ______________________

Date: ______________________

Order:

Description	Order number	Qty.
Reference books:		
Set of the following six books. To order individual copies, use the following order numbers.	SBOF-1627	____
Communications Guide	SC34-0638	____
Extended Address Mode and Performance Analyzer User Guide	SC34-0591	____
Installation and System Generation Guide	SC34-0646	____
Language Reference	SC34-0643	____
Library Guide and Common Index	SC34-0645	____
Messages and Codes	SC34-0636	____
Operator Commands and Utilities Reference	SC34-0644	____
Guides and reference cards:		
Set of the following four books and reference cards. To order individual copies, use the following order numbers.	SBOF-1628	____
Customization Guide	SC34-0635	____
Event Driven Language Programming Guide	SC34-0637	____
Operation Guide	SC34-0642	____
Problem Determination Guide	SC34-0639	____
Language Reference Card	SX34-0165	____
Operator Commands and Utilities Reference Card	SX34-0164	____
Conversion Charts Reference Card	SX34-0163	____
Reference Card Envelope	SX34-0166	____
Set of three reference cards and storage envelope. (One set is included with order number SBOF-1627)	SBOF-1629	____
Binders:		
3-ring easel binder with 1 inch rings	SR30-0324	____
3-ring easel binder with 2 inch rings	SR30-0327	____
Standard 3-ring binder with 1 inch rings	SR30-0329	____
Standard 3-ring binder with 1 1/2 inch rings	SR30-0330	____
Standard 3-ring binder with 2 inch rings	SR30-0331	____
Diskette binder (Holds eight 8-inch diskettes.)	SB30-0479	____

Publications Order Form

Cut or Fold Along Line

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

IBM Corporation
1 Culver Road
Dayton, New Jersey 08810

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

International Business Machines Corporation

IBM Series/1 Event Driven Executive
Operator Commands and Utilities Reference
Order No. SC34-0644-0

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SC34-0644-0
Printed in U.S.A.

Reader's Comment Form

Cut or Fold Along Line

Fold and tape	Please Do Not Staple	Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS	PERMIT NO. 40	ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Information Development, Department 28B
P.O. Box 1328
Boca Raton, Florida 33432

Fold and tape	Please Do Not Staple	Fold and tape

IBM®

IBM Series/1 Event Driven Executive
Operator Commands and Utilities Reference
Order No. SC34-0644-0

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SC34-0644-0
Printed in U.S.A.

Reader's Comment Form

Cut or Fold Along Line

Fold and tape　　Please Do Not Staple　　Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS　　PERMIT NO. 40　　ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Information Development, Department 28B
P.O. Box 1328
Boca Raton, Florida 33432

Fold and tape　　Please Do Not Staple　　Fold and tape

IBM®

IBM®

International Business Machines Corporation

SC34-0644-0

SC34-0644-0
Program Numbers: 5719-XS5,
5719-XX6, 5740-LM6, 5719-LM9
File Number: S1-32
Printed in U.S.A.