IBM / Technical Newsletter

This Newsletter No. SN34-0684
Date February 13, 1981
Base Publication No. SC34-0313-2
File No. S1-32
Previous Newsletters None

IBM Series/1
Event Driven Executive
Utilities, Operator Commands, Program Preparation, Messages and Codes
Program Numbers: **5719-LM5** **5719-LM6** **5719-AM3**
5719-UT3 **5719-UT4**
5719-XS1 **5719-XS2**
5719-XX2 **5719-XX3**
5740-LM2 **5740-LM3**

This Technical Newsletter provides replacement pages for the subject publication. Pages to be inserted and/or removed are:

iii, iv	155, 156	311, 312
3, 4	167 through 170	339 through 342
7, 8	205 through 208	343
17	208.1 through 208.8 (added)	343.1 through 343.12 (added)
17.1, 17.2 (added)	217	344
18	217.1, 217.2 (added)	347 through 350
19, 20	218	350.1 through 350.14 (added)
20.1, 20.2 (added)	235 through 240	413, 414
23, 24	243 through 248	421 through 434
59 through 68	249	435
68.1, 68.2 (added)	249.1, 249.2 (added)	435.1 through 435.24 (added)
81 through 90	250	436
95 through 100	273, 274	436.1 through 436.36 (added)
149, 150	295, 296	

A technical change to the text or to an illustration is indicated by a vertical line to the left of the change.

Summary of Amendments

Corrections and editorial changes were made throughout the book.

Note. Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Systems Publications, Department 27T, P.O. Box 1328, Boca Raton, Florida 33432

SUMMARY OF AMENDMENTS

Operator Commands

The following operator commands have been modified to include support for the IBM Series/1 4969 Magnetic Tape Subsystem:

- $C
- $VARYOFF
- $VARYON

The $VERIFY utility has been added for the Indexed Access Method Version 1 Modification Level 2.

Session Manager

The following changes have been made to the Session Manager for the Event Driven Executive Version 2 (5719-UT4):

- $PL/I (Option 10) has been added to the Program Preparation secondary option menu to support the PL/I compiler.
- Option 3 "Disk Utilities" of the primary option menu has been changed to "Data Management".
- $TAPEUT1 (Option 10) has been added to the "Data Management" secondary option menu to support tape management.

$TAPEUT1 Utility

This new utility is described in Chapter 4. The tape READ/WRITE return codes are described in Chapter 6.

$PREFIND Utility

This utility has been updated to include support for locating tape data sets.

$JOBUTIL - Job Stream Processor

This utility has been updated to support the PL/I compiler.

Suggested Utility Function Table

This table has been expanded to include all the utility programs within this book for easy reference to their functions and commands.

Glossary

New terms have been added to the glossary.

Reorganization

The book has been reorganized. It is now divided into six chapters with an introduction for each chapter.

Chapter 4 presents the utilities in alphabetic order.

Most utility programs are used interactively from a terminal. After a utility program is invoked, you can list its defined operations and command by entering a question mark in response to the 'COMMAND (?):' prompt.

In Chapter 4, the utility programs are presented in alphabetic order along with examples of their usage.

The session manager groups the utility programs by function. The following represents the functional groupings of the utilities along with the operations they perform.

Text Editing Utilities

The text editing utilities provide facilities for entering and editing source programs as follows:

$EDIT1 A line editor that uses host data sets

$EDIT1N A line editor that uses Series/1 data sets

$FSEDIT A full screen editor that uses Series/1 or host data sets

Program Preparation Utilities

The program preparation utilities aid in:

$COBOL Compiling COBOL programs

$EDXASM Compiling Event Driven Language programs

$EDXLIST Reformatting $EDXASM listings

$FORT Compiling FORTRAN programs

$LINK Link editing more that one program together

$PL/I Compiling PL/I programs

$PREFIND Prefinding data sets and overlay programs to shorten program loading time

$S1ASM Assembling Series/1 assembler language programs

$UPDATE Converting an object program into an executable load module

$UPDATEH Converting a host object program into an executable load module

Data Management Utilities

The data management utilities aid in:

$COMPRES Compressing disk or diskette libraries

$COPY Copying disk or diskette data sets or volumes

$COPYUT1 Copying data sets and volumes with dynamic allocation of the receiving data sets

$DASDI Initializing, formatting, and verifying disks or diskettes

$DISKUT1 Allocating and deleting data sets; listing directory data

$DISKUT2 Patching and dumping data sets; listing the error log data set

$DISKUT3 Performing data management functions from another program. $DISKUT3 is described in the System Guide.

$IAMUT1 Building and maintaining Indexed Access Method data sets

$INITDSK Initializing and verifying a direct access storage volume for use with the Event Driven Executive

$MOVEVOL Transferring volumes of data between systems and creating backup copies of an online data base

$PDS Organizing and accessing partitioned data sets from another program. $PDS is described in the System Guide.

$TAPEUT1 Allocating tape data sets, copying data sets or volumes from disk or diskette to tape, from tape to disk or diskette, or from tape to tape, and changing tape attributes.

$VERIFY Verifying and displaying information about indexed data sets

MESSAGES AND CODES

While using the Event Driven Executive, you may encounter return codes, completion codes, and messages. They are found in Chapter 6. Messages and Codes.

HARDCOPY FUNCTION FOR THE 4978/4979 DISPLAY

Pressing the PF6 key or the assigned hard-copy key on the 4978 or 4979 keyboard causes the entire display (24 lines) to be transferred to the designated hard-copy device. (During system generation, the TERMINAL statement is used to define the hard-copy device.) If the hard-copy device has not been defined or is currently busy with another operation, then no action is taken. Otherwise, the screen cursor moves to each line as it is printed, returning to its original position after the page is printed. The hard-copy function should not be activated while the screen is being changed or when I/O is being directed to the screen. Also, while the hard-copy function is in progress, keys (such as the attention key, PF keys, or ENTER) should not be pressed. Simultaneous operation of I/O and the hard-copy function can result in unpredictable results.

$L - Load Program

The $L command loads a program from disk or diskette and starts it.

Syntax

```
$L          program,volume,storage data set(s)
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
volume	The name of the volume where the program being loaded is stored
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)
data set(s)	Data set(s) to be passed to the program being loaded (if specified in PROGRAM statement); specify the data set(s) in the order the program expects.

$L Example

Load a program and pass a single data set:

```
> $L      PROCESS,EDX003 MYDATA
```

This example shows the command and parameters entered in single line format. It is good practice to enter as much of the required information as possible on the same line of input as the $L command to minimize the time the loader is busy.

Note: Wait until the system is initialized before loading a program. If your system has timers, the system is initialized when the 'SET TIME AND DATE USING $T' appears (or when the time and date are printed). If your system does not have timers, the system is initialized when it enters the wait state after the storage map has been displayed.

This page intentionally left blank.

$P - Patch Storage

Allows main storage to be patched online. Enter the patch data in response to prompting messages.

Syntax

```
$P          origin,address,count
Required:   origin,address,count
Default:    None
```

Operands	Description
origin	The hexadecimal origin address (program load point).
address	The hexadecimal address in the program at which the patch is to start.
count	The decimal number of words to patch.

Example - Patch word X'100' of program loaded at 0 to X'FFFF'

```
> $P
ENTER ORIGIN: 0000
ENTER ADDRESS,COUNT: 0100,1
0100: C462
DATA: FFFF
ANOTHER PATCH? N
```

$T - Set Date and Time

Enters a new date and time into the system and resets the realtime clock. You can only use $T from terminals having the label $SYSLOG and $SYSLOGA. After entering the time, the timer is started at the instant carriage return/ENTER is pressed. This resets the seconds to zero.

Notes:

1. Make sure your time and date entry is correct as the system does not verify this data.

2. If $T is entered from other than $SYSLOG or $SYSLOGA, it is equivalent to entering $W.

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.
time	The current time.

Example - Set date and time

```
> $T
DATE(M.D.Y): 8:22:79
TIME(H.M): 11:15
```

$VARYOFF - Set Device Offline

Sets the status of a disk, diskette, diskette magazine unit, or tape drive to offline.

On the 4966 diskette magazine unit, each diskette volume in individual diskette slots or either of the diskette magazines can be set to offline.

When you vary a tape device offline, that tape drive is rewound to the load point and set logically offline.

Syntax

```
$VARYOFF     ioda slot
Required:    ioda
Default:     None
```

Operands	Description
ioda	The hexadecimal device address of the device to be varied offline.
slot	The slot number of the diskette to be varied off-line; this parameter applies to the 4966 only. 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

Examples:

Vary offline the volume in slot 2 of a 4966 device at address 22

```
> $VARYOFF    22 2
IBMIRD OFFLINE
```

This page intentionally left blank.

```
$VARYON      ioda slot|file 'EX'
Required:    ioda
Default:     file defaults to 1, maximum value of 255
```

Note: The OR symbol (|) indicates mutually exclusive operands.

Operands	Description
ioda	The hexadecimal device address of the device to be varied online.
slot	The slot number of the diskette to be varied online; this parameter applies to the 4966 only. 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
file	The decimal file number on the tape to be accessed. This parameter applies to the tape drive only.
'EX'	This parameter applies to tape only and requests an expiration date override. If a tape data set is initialized with an expiration date, this parameter must be used to write to that tape data set and the file number must be specified.

Examples:

Vary diskette in slot 1 of 4966 at device address 22 online

```
> $VARYON 22 1
IBMIRD ONLINE
```

Vary a standard label (SL) tape (volume 123456) at address 4C online and access the first file.

```
> $VARYON 4C
123456 ONLINE
```

Vary a non-labeled (NL) tape at address 4C online and access the second file, where TAPE1 was the ID assigned at system generation.

```
> $VARYON 4C 2
TAPE1 ONLINE
```

Vary a standard label tape at address 4D online. The first file of this tape has an expiration date that has not expired; however, output to this file is allowed.

```
> $VARYON 4D 1 EX
OVERRIDE EXPIRATION DATE CHECK? (Y,N): Y
123456 ONLINE
```

$COPY - COPY DATA SET

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

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

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

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

When copying program members, the size of the target member must be equal to or greater than the source member. When copying data members, an entire member may be copied, or only a selected number of records (partial copy) may be copied. If the entire member is to be copied, 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 being copied from the source member.

When copying diskette volumes to disk, the target data set must be of equal or greater size than the diskette size in records. When copying a disk volume to another disk volume, 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. If the source volume is smaller than the target, you are prompted for the name of the target data set into which the source volume will be copied.

$COPY

Invoking $COPY

$COPY is invoked by the $L command or primary option 3 of the session manager.

$COPY Commands

The commands available under $COPY are listed below. To display this list at your terminal, enter a question mark in respond 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 the commands are displayed, you are again prompted with COMMAND (?):. You respond with the command of your choice (for example, CD).

Absolute Record Copy

$COPY provides an absolute record capability, using the special system names $$EDXLIB and $$EDXVOL. This allows you to copy a record relative to the beginning of a volume ($$EDXVOL) or relative to the beginning of a library ($$EDXLIB). This capability can be used when copying one diskette volume to another. The CV command of $COPY does not copy the first cylinder on diskette. If the source diskette is an IPL volume (has IPL text and $EDXNUC), the IPL text, contained in the first record of the first cylinder, is not copied to the target diskette. Therefore, the target diskette volume, although containing a supervisor in $EDXNUC, is not able to load that supervisor when the IPL key is pressed.

To copy the IPL text to the target diskette, use the CD command of $COPY; specify $$EDXVOL as the data set name, and record 1 as the first and last record to be copied.

See the data set naming conventions in the &sys. for a description of the special data set names, $$EDXVOL and $$EDXLIB, which are used when doing absolute record and basic exchange copying.

CD - Copy Data Set

The CD command is used to copy disk or diskette data sets to a preallocated disk or diskette target data set. When copying data sets, an entire data set may be copied or only a select number of records (partial copy) may be copied. If the entire data set is to be copied, 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 being copied from the source data set.

CD Examples

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. Data sets allocated as program organization may not be copied to a data set allocated as data organization.

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

$COPY

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 to be copied from the source data set, the following message is issued:

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

When the output data set is on disk or diskette, the end of data pointers are updated.

CV - Copy Volume

The CV command allows you to copy entire volumes, providing a volume backup capability. A disk volume may be copied to another disk volume, a diskette volume to another diskette volume, or a diskette volume to a preallocated disk data set of appropriate size in records, as follows:

Type	Number of Records at 128 Bytes/Sector
Diskette 1	949
Diskette 2	1924

Volume copy operations do not add the members in a source volume to the target volume; the original contents of the target volume are replaced, including the directory.

If you have two or more 4964 Diskette units, or a 4964 and a 4966 Diskette Magazine unit, diskette volume copies between diskette devices are possible. If you have a single diskette drive and a disk, diskette volume copies can be performed using the following procedure:

1. Allocate a target data set on 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 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.

$COPYUT1

CV 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 (?):
```

The CV command copies the entire diskette volume. Therefore, the target data set should be of equal or greater size than the diskette 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.

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 source is copied 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 original contents of the target volume are replaced, including the directory. As a result, the original contents of the target disk volume are no longer accessible.

Note: See the System Guide for an explanation of disk and diskette organization.

RE - Copy from Basic Exchange

The RE command copies a basic exchange data set from a diskette to a disk data set. A basic exchange data set is contained on a diskette that was formatted for the Standard for Information Interchange. Only one-sided, 128-byte diskettes can be used because EDX recognizes only one volume on a diskette. The target disk data set must be allocated using $DISKUT1 before using the RE command.

You are prompted for the source diskette data set name and volume, the target disk data set name and volume, the number of the first record to be written to the target data set, and the basic exchange data set name.

RE Example

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 (?):
```

$COPYUT1

Copy Basic Exchange Data Set to Disk: In this example, 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 (?):
```

WE - Copy to Basic Exchange

The WE command copies a disk data set to a basic exchange data set on diskette. The diskette data set must be allocated before using WE. 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.

You are prompted 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.

WE Example

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 (?):
```

Copy a Disk Data Set to a Basic Exchange Diskette: In this example, the beginning and ending records numbers on 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 the diskette contains uninitialized HDR1s. Data on the diskette is read and written two sectors per I/O operation.

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

$DASDI

$DASDI - FORMAT DISK OR DISKETTE

$DASDI initializes your 4962 or 4963 disk or formats diskettes on the 4964 or 4966 diskette units. The utility can be used online. When this utility is invoked, you are prompted for one of the following disk or diskette initialization options:

- Option 1 - 4964, 4966 diskette initialization.
- Option 2 - 4962 disk initialization
- Option 3 - 4963 disk initialization

$DASDI may be loaded into any partition, but $DASDI will load initialization routines into partion 1. ($IDSKETT, $I4962, $I4965) into partition 1. An error message will be returned if partition 1 does not have space for initialization routines.

Caution: For disk initialization, a program that accesses the disk being initialized should not be run concurrently with this utility.

Diskette initialization can run concurrently with other programs.

Invoking $DASDI

$DASDI is invoked by the $L command or primary option 3 of the session manager.

When $DASDI is invoked, you are prompted for one of the following initialization options:

- Option 0 - Stand-Alone Dump 4964/4966 Diskette Initialization
- Option 1 - 4964, 4966 Diskette Initialization
- Option 2 - 4962 Disk Initialization
- Option 3 - 4963 Disk Initialization

Notes:

1. $DASDI must be loaded in partition 1.
2. When options 2 and 3 are executing, do not run a program that accesses the disk being initialized.

3. Diskette initialization can run concurrently with other programs.

Option 0 - Create a Stand-Alone Dump Diskette 4964/4966

Option 0 uses a 4964 or 4966 diskette unit to initialize a 2-sided single density 128-byte diskette to be used for stand-alone dumps. $DASDI loads a program that places IPL text and the stand-alone dump utility on the front of the diskette. Once the diskette is created, it is ready for use.

This diskette is reusable and does not have to be reformatted or recreated after it has been used to take a stand-alone dump.

Option 0 Example

Create a Stand-Alone Dump Diskette on a 4966:

```
> $L $DASDI
$DASDI         15P,00:28:55, LP= 7E00

DIRECT ACCESS DEVICE INITIALIZATION
  DISK INITIALIZATION OPTIONS:
    0 = CREATE STAND-ALONE DUMP DISKETTE 4964/4966
    1 = 4964, 4966 DISKETTE INITIALIZATION
    2 = 4962 DISK INITIALIZATION
    3 = 4963 DISK INITIALIZATION
    4 = EXIT DISK INITIALIZATION
  ENTER DISK INITIALIZATION OPTION: 0

*********************************************
*          DISKETTE FORMATTING PROGRAM      *
*  IF FORMATTING IS IN PROGRESS, DO NOT     *
*  CANCEL ($C) THIS PROGRAM.  INSTEAD,      *
*  ENTER ATTN/$DASDI   TO FORCE TERMINATION. *
*********************************************

ENTER DISKETTE ADDRESS IN HEX: 22

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

IBMEDX VARIED ONLINE
FORMATTING COMPLETE
STAND-ALONE DUMP DISKETTE CREATED
ANOTHER DISKETTE? N
DASDI   ENDED AT 00:32:12
```

Option 1 - 4964, 4966 Diskette Initialization

Diskette Formats

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

1. Format for use with the Series/1 Event Driven Executive

2. Format to the IBM Standard for Information Interchange

3. Format entire diskette to 128, 256, or 512 byte records.

If you select the Event Driven Executive format, all tracks are formatted for 128 byte sectors. Also, cylinder 0 is formatted according to the IBM Standard for Information Interchange. The assigned volume label is IBMEDX.

Note: Use this format if all cylinders are to be formatted to 128-byte sectors.

If you initialize according to the IBM Standard for Information Interchange, Cylinder 0 is formatted for 128-byte sectors, and the remaining cylinders are formatted for either 128-, 256-, or

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
$DASDI   ENDED AT 01:01:27
```

Note: The fixed head area on the 4963 is always referred to as Cylinder 511. This cylinder contains eight heads (16 through 23) and 64 sectors (zero through 63).

$DEBUG - DEBUGGING TOOL

$DEBUG is a tool for locating errors in programs. By operating a program under control of $DEBUG, you can:

- Stop the program each time execution reaches any of one or more instruction addresses that you have specified. These addresses are known as breakpoints.

- List the contents of specified storage locations or register contents each time the program execution reaches one or more of your breakpoints.

- 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. Optionally, program execution can be stopped after each instruction is executed within a trace range. Also, optionally, storage locations or register contents can be displayed on the terminal after the execution of each instruction within a trace range.

- Restart program execution at the breakpoint or trace range address where it is currently stopped. Or, in the case of Event Driven Language instructions, 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.

Using these functions, 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.

To use $DEBUG effectively, you must have a printed listing of the program to be debugged which shows the storage addresses of each instruction and data area of interest. To obtain such a listing, specify PRINT GEN in the source program, after the PROGRAM statement, at assembly time. A PRINT NOGEN should precede the PROGRAM statement to prevent the unnecessary printing of many system EQU statements, etc. For $EDXASM a satisfactory listing is produced by specifying LIST.

Debug Usage Considerations

The program debug facility aids in testing multitasked programs in a multiprogramming and multiuser environment. All of your interactions are via terminals and do not require the use of the machine console. A summary of the major features of the $DEBUG program follows:

Notes:

1. If the program being debugged is ended, $DEBUG must be ended.
2. $DEBUG should be invoked from a terminal other than the one used by the program to be tested if the program uses 4978/4979 terminals in STATIC screen mode.
3. Multiple breakpoints and trace ranges can be established.
4. Several users can each use separate copies of $DEBUG concurrently, if sufficient storage is available.
5. Series/1 assembler language as well as Event Driven Language instructions can be traced and tested.
6. Both supervisor and application programs can be debugged.
7. Task names are automatically obtained from the program to be tested.
8. Task registers #1 and #2 can be displayed and modified.
9. Hardware registers R0 through R7 and the IAR can be displayed and modified.
10. The task return code words can be displayed and modified.
11. Five different data formats are accepted by the list and patch functions.
12. No special preprocessing of a program is required to permit it to be debugged.
13. All address specifications are made as shown in the program assembly listing without concern for the actual memory addresses where the program is loaded into storage for testing.
14. No processing overhead is incurred unless the hardware trace feature is enabled. Even then, the hardware trace feature is only enabled for specific tasks.

15. The debug facility can be activated for a program that is experiencing problems but was previously loaded without the debug facility.

16. A program can be debugged by loading $DEBUG from a terminal other than the one from which the program to be tested was loaded.

17. Breakpoints or trace ranges specified during a debug session can be listed.

18. $DEBUG can control the execution of programs containing up to 20 tasks.

The $DEBUG program can be used to test different types of programs. The most common usage is to debug application programs written using the Event Driven Language instruction set. However, it can also be used to test portions of application programs that are written in assembler language and portions of the supervisor program that are written in either Event Driven Language or Series/1 assembler language. Testing of the supervisor should normally be required only if you are making your own modifications or additions to this program.

You can use $DEBUG to debug overlay programs by loading the primary program that will subsequently load the overlay program to be debugged. Load $DEBUG after the overlay program is in storage. (For more information on debugging overlay programs that are part of the Event Driven Language compiler, $EDXASM, refer to the Internal Design). To suspend execution of the overlay program so that $DEBUG can be loaded, enter a READTEXT or QUESTION as the first instruction of the overlay program. Multiple Terminal Manager users should code a CALL ACTION instruction to provide the required function. When the overlay program is entered, it pauses at the first instruction and waits for input. At this point, load $DEBUG. This can be done from another terminal assigned to the same partition. Specify the overlay program name when prompted for the program name and indicate that no new copy of the overlay program is to be loaded.

The $DEBUG utility can then be used to set breakpoints and perform other functions as required. If the overlay program causes a program check, it is cancelled by the system. If an overlay program terminates through a PROGSTOP or for any other reason and is reloaded by the primary program, any breakpoints or patches made prior to the termination are lost.

Use of certain capabilities of $DEBUG requires a thorough knowledge of both the supervisor and debugging techniques. For example, altering the contents of storage locations occupied by the supervisor or contents of the Series/1 hardware registers could have undesirable effects on the operation of the supervisor or application programs in operation concurrently with $DEBUG.

Note: Only those instructions that execute as part of a task can be debugged. Those portions of the supervisor program that service interrupts created by various hardware devices (disk, timers, terminals, etc) cannot be executed under control of $DEBUG.

Start and Termination Procedure

The primary method for activating the debug facility is to load $DEBUG and then specify the name of the program to be tested, when prompted (DBUGDEMO in the following example). $DEBUG then loads your program, inserts a breakpoint at the first executable instruction, and notifies you that your program is stopped at this point. For example:

```
> $L $DEBUG
$DEBUG        26P,09:10:17, LP=5200
PROGRAM NAME: DBUGDEMO
DBUGDEMO       4P,09:10:28, LP=6700
DBUGDEMO  STOPPED AT  009E
```

$DEBUG

$DEBUG Commands

The following commands are available:

```
AT        - Set breakpoints and trace ranges
BP        - List breakpoints and trace ranges
            thus far specified
END       - Terminate debug facility
GO        - Activate stopped task
GOTO      - Change execution sequence
HELP      - List debug commands
LIST      - Display storage or registers
OFF       - Remove breakpoints and trace ranges
PATCH     - Modify storage or registers
POST      - Post an event or process interrupt
PRINT     - Direct output to another terminal
QUALIFY   - Modify base address
WHERE     - Display status of all tasks
```

How To Enter A Command

A command is entered by pressing the ATTENTION key on your terminal and entering the command name, or the command name plus the required parameters for the command, in response to the prompting message '>'.

Syntax Summary

In the command syntax examples and descriptions in the following sections, keyword parameters are capitalized and variable parameters are shown in lower case. Whenever one of several keywords can be chosen, these keywords are separated by a slash(/). The examples show the various formats of each command which are available for different purposes. Detailed syntax descriptions are presented under $DEBUG Command Descriptions.

Set breakpoints and trace ranges:

```
AT ADDR address NOLIST/LIST NOSTOP/STOP
AT TASK taskname start-add end-add EDX/ASM NOLIST/LIST
       NOSTOP/STOP
AT ALL NOLIST/LIST NOSTOP/STOP
AT *   NOLIST/LIST NOSTOP/STOP
```

Terminate $DEBUG:

```
END
```

Note: When the program being debugged is ended, $DEBUG must be ended.

Activate breakpoints or trace ranges:

```
GO ADDR address
GO TASK taskname start-add end-add
GO ALL
GO *
```

List $DEBUG commands:

```
HELP
```

Display storage or registers:

```
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 *
```

Remove breakpoints or trace ranges:

```
OFF ADDR address
OFF TASK taskname start-add end-add
OFF ALL
OFF *
```

$DEBUG

Modify storage or registers:

```
PATCH ADDR address        length mode
PATCH R0...R7   taskname  length mode
PATCH #1/#2     taskname  length mode
PATCH IAR       taskname  length mode
PATCH *
```

Post events or process interrupts:

```
POST ADDR address code
POST PIxx code
POST *
```

Direct output to another terminal:

```
PRINT  terminal name
PRINT  *
PRINT
```

Modify base address:

```
QUALIFY base disp
Q base displ
```

Display status of all tasks:

```
WHERE
```

Display breakpoints and trace ranges:

```
BP
```

Change execution sequence:

```
GOTO  current-address  new-address
```

Example: The command syntax permits most keyword parameters to be abbreviated to a single character, except ALL which conflicts with ADDR; entry of AL for ALL and A for ADDR is permitted. You are prompted for command parameters individually, unless you are sufficiently familiar with the syntax to enter a complete command on a single line. For example, to set the task TIMET into a program trace between addresses 0 and 3000 and also print the contents of both task registers in task LOOP2 in decimal mode but continue processing, the following interactive keyboard sequence may occur. Each response is terminated by a RETURN key entry.

```
> AT
OPTION(*/ADDR/TASK/ALL): TASK
FIRST ADDRESS: 0600
LAST ADDRESS: 1000
TRACE TYPE(EDX/ASM): EDX
LIST/NOLIST: N
STOP/NOSTOP: N
1 BREAKPOINT(S) SET
```

Identical results are obtained by entering the single response:

```
> AT T TIMET 0600 1000 E N
      1 BREAKPOINT(S) SET
```

$DEBUG Command Descriptions

AT - Set Breakpoints and Trace Ranges

AT sets breakpoints and trace ranges. The LIST and STOP options established for a breakpoint or trace range are executed prior to executing the instruction that satisfied the breakpoint or trace range specification. When the specification for a breakpoint or trace range is satisfied, the task currently active is detoured and $DEBUG performs the following actions for the subject task: prints its status, prints the LIST specification, and optionally puts the task into a wait state. If the NOSTOP option was requested, task status is printed as "taskname CHECKED AT XXXX". The STOP option generates a "taskname STOPPED AT XXXX" message.

The LIST and STOP options for all currently defined breakpoints and trace ranges can be modified by entering AT ALL. Similarly, the specifications for the most recently entered AT command can be altered by the AT * option.

Notes:

1. You cannot set breakpoints in ATTNLIST routines.

2. You can only set breakpoints on EDL instructions within an EDL program.

Syntax

```
AT ADDR address NOLIST/LIST NOSTOP/STOP
AT TASK taskname start-add end-add EDX/ASM 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 a breakpoint is to be inserted.

LIST - Display Storage or Registers

LIST displays the contents of memory locations, or task registers, or hardware registers, or the IAR. The LSB can be displayed by listing the IAR with a length of 11 words. Any register data is only guaranteed to be current if the corresponding task is stopped by a $DEBUG breakpoint or trace range. To repeat the most recently specified LIST command or to verify (list) a patch you have just entered, use "LIST *".

Syntax

```
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 *
```

Operands	Description
ADDR	Keyword indicating this is a display of a storage location.
address	Address of the storage location to be displayed.
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
R0/.../R7	One of the Series/1 hardware registers R0 through R7. To define the start of the LIST.
taskname	Name of task from which register data is to be displayed. For programs containing only a single task, omit this parameter.
#1/#2	Task register #1 or #2 specification.

$DEBUG

IAR	Keyword indicating the IAR (Instruction Address Register) is to be displayed.
*	The most recently specified LIST specification is to be used. This is determined by the last usage of a LIST or PATCH command.
TCODE	Keyword indicating the task return code word(s) (first two words of the TCB) to be displayed.

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. To resume normal operations after removing trace requests, the task must be stopped at a breakpoint or trace range. If a task is currently stopped at the requested breakpoint or trace range, this task is automatically reactivated.

Syntax

```
OFF ADDR address
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 a breakpoint has previously been established.
TASK	Keyword indicating a trace range is to be removed.
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	All breakpoints and trace ranges are to be removed.
*	The most recently referenced breakpoint or trace range specification is to be used. This specification is determined by the last usage of an AT, GO, or OFF command.

$DEBUG

PATCH - Modify Storage or Registers

PATCH modifies the contents of memory locations, task registers, hardware registers, and the IAR (Instruction Address Register). The entire LSB (Level Status Block) can be modified by patching the IAR with a length specification of 11 words. The patch to any register data is only guaranteed if the corresponding task is inactive or stopped by a $DEBUG breakpoint or trace range. To respecify the data for the most recent patch or to patch the data displayed by the most recent LIST command, enter 'PATCH *'.

After the patch command is entered, the current memory or register content is displayed, and you are prompted for the patch data (a string of data entries that satisfy the length and mode specifications). The entries are separated by spaces, for example, data...data. After the patch data is entered, you can apply the patch by responding YES, abort by responding NO, or indicate additional patches with a CONTINUE reply to the prompting message. By specifying CONTINUE, the patch is performed and prompting continues for entry of new length, mode, and data specifications to memory or register locations immediately behind your previous patch.

If less data than specified with the length operand is entered, the effective patch is padded with blanks for character data and zeros for all other data types.

Syntax

```
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 *
```

Operands	Description
ADDR	Keyword indicating this is a storage patch.
address	Address of the storage location to be modified.
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
R0/.../R7	One of the Series/1 hardware registers R0 through R7, where the patch is to be started.
taskname	Name of task for which register data is to be modified. For programs containing only a single task, omit this parameter.
#1/#2	Task register #1 or #2 specification.
IAR	Keyword indicating the IAR (Instruction Address Register) is to be modified.
TCODE	Keyword indicating the task return code word(s) (first two words of the TCB) to be modified
*	The most recently referenced LIST or PATCH specification is to be used and is determined by the last usage of a LIST or PATCH command

$DEBUG

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) to be posted is contained in the second word of a WAIT command as shown on a program assembly listing. Process interrupts can also be posted by name, using the PIxx option.

Syntax

```
POST ADDR address code
POST PIxx code
POST *
```

Operands	Description
ADDR	The address of an ECB (Event Control Block).
address	ECB address to be posted.
code	Decimal code to be posted to the specified ECB.
PIxx	Name of process interrupt PI1 to PI99.
*	The most recently referenced ECB address or PIxx name and code specification is to be used.

PRINT - Direct Output to Another Terminal

PRINT allows you to direct $DEBUG output to the specified terminal. For example, the output of a LIST command would appear on the specified terminal.

Syntax

```
PRINT  terminal name
```

Operands	Description
terminal name	The terminal to which the output is to be directed

SS - Set Program Storage Parm: The following example shows reducing the dynamic storage to be allocated for the COBOL compiler at program load. The SS command requires the size in bytes to be expressed in decimal. The value requested, if not an even multiple of 256, will be rounded up.

```
> $DISKUT2

USING VOLUME EDX002
COMMAND (?): CV ASMLIB

COMMAND (?): SS $COBOL
ENTER NEW STORAGE SIZE IN BYTES: 2816
OLD STORAGE SIZE WAS 8448
OK TO CONTINUE? Y

COMMAND (?): EN
$DISKUT2 ENDED AT 08:36:02
```

If an invalid value is entered, a message is issued stating the value must be in the range of 0 to a maximum number of 64K, minus the program size.

$DIUTIL - DISPLAY DATA BASE UTILITY

$DIUTIL maintains the disk resident data base used with graphics applications. 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 data base due to deleted members
- Display contents of data base
- Copy data base
- Copy individual members of data base
- Allocate and build a data member

This utility is normally used only when no other programs of the Display Processor are in use. The online data base can be changed or you may select another data base to be referenced. 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

To start execution of $DIUTIL:

1. Load the program $DIUTIL specifying the appropriate data set. $DIFILE, the online data set, or any other data set can be used. However, you should make sure that another user or program is not changing or using the same data set.

2. The system responds with the Program Loaded message followed by:

```
DISPLAY DATA BASE UTILITY
COMMAND (?):
```

CM - Copy Member

Copies a member from the source data base to the target data base. The options available in MD are also included in CM.

```
COMMAND (?): CM
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
ENTER MEMBER NAME TO BE COPIED
PLOT
PLOT     COPIED
COPY COMPLETED

COMMAND (?):
```

The extents of the data set are not changed if a modified member can fit in the currently allocated space. If the modified member is larger than currently allocated space, then the old member is deleted and a new member must be allocated with larger extents. Space must exist in the data set for the copied member.

DE - Delete a Member

Removes display or data members from the data base. You are prompted for the name of the member to be deleted and asked 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

Causes the Display Processor utility to be terminated.

Dump Part of Supervisor Partition to Printer (cont.)

```
STORAGE MAP:          $SYSCOM AT ADDRESS 3420

PART#  NAME       ADDR  PAGES  TCB

 P1   **PART**    9F00    97
 P1   $TRAP       9F00    21   B2B4
 P1   **FREE**    B400    76

 P2   **PART**    0000   256
 P2   **FREE**    0000   256

 P3   **PART**    0000   256
 P3   **FREE**    0000   256

 P4   **PART**    0000   256
 P4   $SMMAIN     0000     4  02F8
 P4   $SMLOG      0400    34  1978
 P4   **FREE**    2600   218
```

$DUMP

Dump Part of Supervisor Partition to Printer (cont.)

```
TERMINAL LIST:

  NAME     CCB    ID   ADDR

$SYSLOG    1876 0406 0004
$TERM1     1A82 040E 0005
$TERM2     0C32 040E 0006
$SYSLOGA   1DCC 0010 0000
$SYSPRTR   1F86 0306 0001

DSK(ETTE) LIST:

NAME     DDB  TYPE     ID      ORG     SIZE      LIB    ADDR

EDX001   165E PRI     0106       0       75       27    0002
EDX002   16F0 PRI     00AA       0       92      241    0003 IP
ASMLIB   1782 SEC                92       16        1
SUPLIB   17A2 SEC               108       16        1
MACLIB   17C2 SEC               124       78        1
EDX003   17E2 SEC               202      102        1

SUPV BEGINNING AT ADDRESS 0000 FOR 139 PAGES


0000    6802 882A 0000 0000 8968 8826 8969 8826    |......
0010    0000 0000 8968 8826 0A76 0A12 8968 8826    |......
   SAME AS ABOVE
0100    12DC 8BC2 0004 0006 1010 6A08 03F8 5B22    |...B..

ANOTHER AREA? (Y/N): N                          Note
```

Note: $DUMP allows you to request several partial dumps. If a 'Y' response is entered, then $DUMP prompts you for the starting and ending addresses that are to be dumped. See note 4.

$EDIT1 AND $EDIT1N - LINE EDITORS

$EDIT1 and $EDIT1N provide a text editing facility (primarily used for source program entry and editing) that can be invoked simultaneously with the execution of other programs. The Host Communication Facility related version ($EDIT1) provides a few commands for data communication using the Host Communications Facility IUP on the System/370 so that almost the entire process of program preparation can be controlled from a Series/1 terminal. The native program preparation version ($EDIT1N) produces members that can be processed by the Series/1 assembler.

Both versions work with 80-character lines that are line numbered in positions 73-80 and are invoked by the $L command.

Data Set Requirements

One work data set is required by the editing facility and must be allocated on disk or diskette using $DISKUT1. You are prompted for its name when either version is loaded. This data set contains both your data and some index information during the editing session, and 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)

The required data set size can be calculated as follows: number of text lines (n) divided by 30, times 11, plus 1 ((n/30 x 11) + 1).

Sequence of Operations

When the edit program is loaded, it prompts you for the name of the work data set to be used. If an existing data set is to be edited, the READ command should be used to copy the data set to the work data set. For a new data set, edit mode should be invoked. The contents of the work data set can be printed using the LIST command.

The EDIT command is used to enter edit mode. Edit subcommands are then recognized until terminated by the END command.

Note: You should use the VERIFY ON subcommand until you become familiar with the editing process.

The TABSET subcommand is used, if desired, to specify the tab character and tab column. This eliminates the entry of blanks when a substantial amount of the text to be entered is in tabular format or begins in a particular column.

Data can be entered 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). Portions of the edited data can be listed at the terminal using the LIST command.

The position of the current line pointer is controlled by the FIND, TOP, BOTTOM, UP, and DOWN subcommands.

Edit mode is terminated with the END command. When the text has been edited, copy the work data set to a permanent data set using either the WRITE or SAVE subcommand. The work data set is in a blocked format that is incompatible with most Event Driven Executive functions. Automatic translation from text editor format to source statement format is performed.

The following figure shows the primary commands and subcommands available under $EDIT1/$EDIT1N.

$FONT - PROCESS 4978 CHARACTER IMAGE TABLES

$FONT creates or modifies the character image tables for the 4978 display station and the 4974 matrix printer. These devices have an image store, containing the bit patterns which, when interpreted by the hardware, result in the display or print of characters on the screen or printer. Each character bit pattern is associated with an EBCDIC code and is defined by a dot matrix that is coded into eight bytes of data. These bit patterns can be changed to alter the appearance of the characters displayed, or if desired, to create entirely new characters.

You can use the 4979 display station to change the character image tables of a 4978 display station or a 4974 matrix printer. However, the DISP command of $FONT will not display the character image for the device being modified. The character set of the 4979 will be displayed.

> **Note:** The image buffer of the 4974 can be restored to the standard 96-character set by using the RE command of $TERMUT2.

Invoking $FONT

$FONT is invoked by the $L command or primary option 4 of the session manager.

$FONT

$FONT Commands

The commands available under $FONT are listed below. To display this list at your terminal, enter a question mark in response to the prompting message COMMAND (?):.

```
COMMAND (?): ?

DISP -- DISPLAY TABLE
EDIT -- ENTER EDIT MODE
SAVE -- SAVE TABLE
PUT  -- LOAD TABLE INTO DEVICE
GET  -- READ TABLE FROM DEVICE
END  -- END PROGRAM

COMMAND (?):
```

After the commands are displayed, you are again prompted with COMMAND (?):. You respond with the command of your choice (for example, GET).

Data Set Requirements

$FONT requires a preallocated data set of 2048-bytes (8 records) for the 4978 or 768-byte (3 records) for the 4974. The data set can contain the character image table, or it can represent storage for a new table to be constructed.

The 4978 and 4974 image stores may be loaded from a disk or diskette data set. With the exception of the 4978, you should allocate an image store data set before using $FONT. Use $DISKUT1 to allocate the data set.

For the 4978, the system-supplied image store data sets $4978ISO or $4978CSO can be used with $FONT. However, these data sets are automatically loaded to every 4978 supported by the supervisor at initialization, and modifications made will be reflected in all the displays. This may not be desirable in all cases.

DISP - Display Table

The DISP command displays all 256 EBCDIC codes, along with the characters that are generated for each code by the associated bit patterns in the image store. However, the 4974 matrix printer supports a 96 character set; therefore, only the 96 support characters are displayed and the unsupported characters are left blank. For descriptions of the characters supported by the 4974 matrix printer and the 4978 display stations, see the 4974 Printer Description and the 4978 General Information manuals.

If the data set to contain the image store was just allocated, and does not contain an image table, a meaningless display will result. To acquire an image table to work with, use the GET command to read an image table from the 4978 or 4974.

DISP Example

This screen display is in compressed format and does not show the entire image table as it would be displayed.

```
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
```

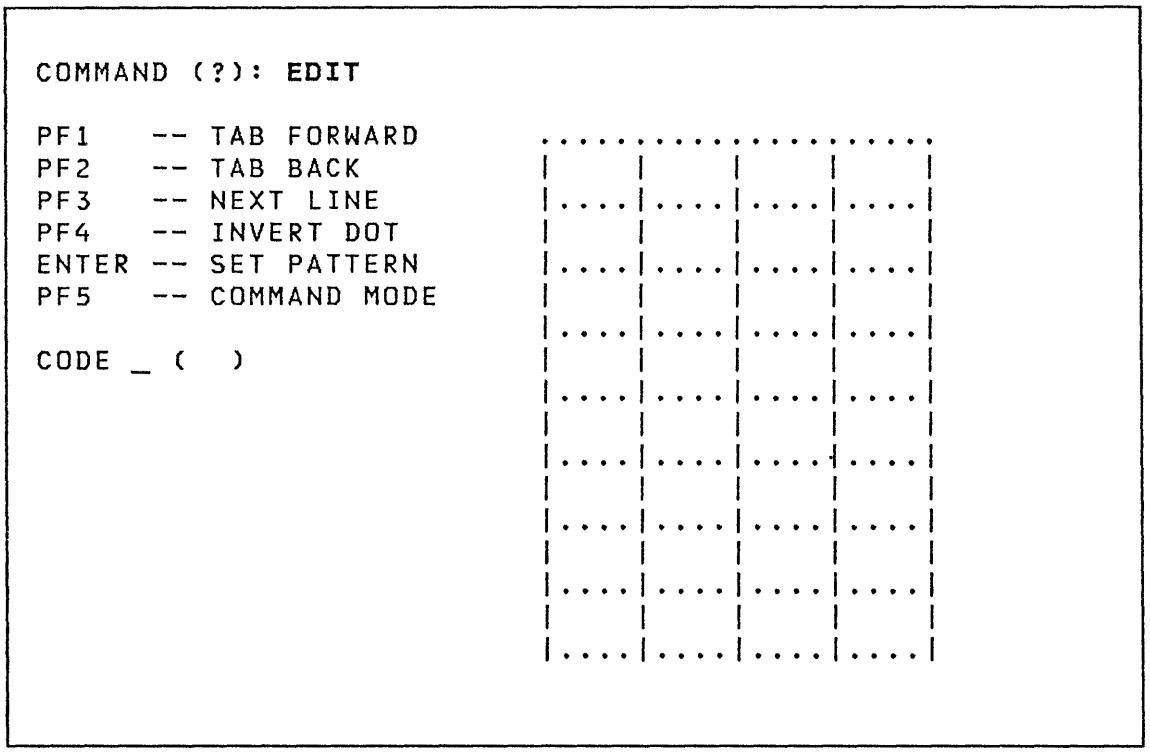
The characters are displayed to the right of the EBCDIC codes with which they are associated. To modify or create a character, EDIT mode must be entered. Display mode is ended by pressing the ENTER key and the COMMAND (?): prompt will appear again. You can then enter EDIT mode.

Note: If you are using a 4979 to change a 4978 or 4974 image store, the DISP command will display the character set of the 4979 and not the current image table of the 4978 or 4974.

$FONT

EDIT - Enter Edit Mode

The EDIT command is used to modify or create characters in the image table. An enlargement of a dot matrix pattern (a 4 x 8 grid) is displayed. This grid is used to create or modify characters within the current image table.



For each grid row, the center of seven overlapping dot areas occur at the centers of each of the four squares and at each of the three interior grid lines.

GET - Read Table from Device

The GET command reads the image store from a 4978 or 4974 into the preallocated work data set and that image store becomes the current table. The GET command followed by a SAVE provides a means for initializing a data set with a character image table.

Syntax

```
GET device

Required: device
Default:  terminal from which you invoked $FONT
```

Operands	Description
device	Name of the device (specified on the TERMINAL statement) whose image table is to be read into the work data set

If you do not specify a device name on the same line as the GET command, you are prompted as follows:

```
TERMINAL NAME:
```

If you do not enter a name following the prompt and press ENTER, GET defaults to the terminal from which you invoked $FONT. If the device is not a 4978, the following message is issued:

```
DEVICE NOT SUPPORTED
```

GET Examples

```
COMMAND (?): GET
TERMINAL NAME: DSPLAY1
```

```
COMMAND (?): GET DSPLAY1
```

PUT - Load Table into Device

The PUT command loads the specified device with the altered or new image table. The image table is only current for the particular device until its image store is loaded with a different image table. Also, the image store will revert to its original state when the system is initialized again or there is a power failure.

Note: Loading a specified device with an altered or new image table can also be accomplished using the LI command (load a control store) of $TERMUT2.

Syntax

```
PUT device

Required: device
Default:  terminal from which you invoked $FONT
```

Operands	Description
device	Name of the device (specified on the TERMINAL statement) whose image store is to receive the current table

If you do not specify a device name on the same line as the PUT command, you are prompted as follows:

```
TERMINAL NAME:
```

If you do not enter a name following the prompt and press ENTER, PUT defaults to the terminal from which you invoked $FONT. If that device is not a 4978, the following message is issued:

```
DEVICE NOT SUPPORTED
```

PUT Examples

```
COMMAND (?): PUT
TERMINAL NAME: DSPLAY1
```

```
COMMAND (?): PUT DSPLAY1
```

SAVE - Save Table

The SAVE command writes the current image table, reflecting any changes made during edit mode, to the data set designated at load time.

Syntax

```
SAVE

Required: none
Default:  SAVE attempts to store the current image table
          in the data set allocated for the 4978
```

If you just specify SAVE, the table is saved for the 4978. If you specify 4974, the table is saved for the 4974.

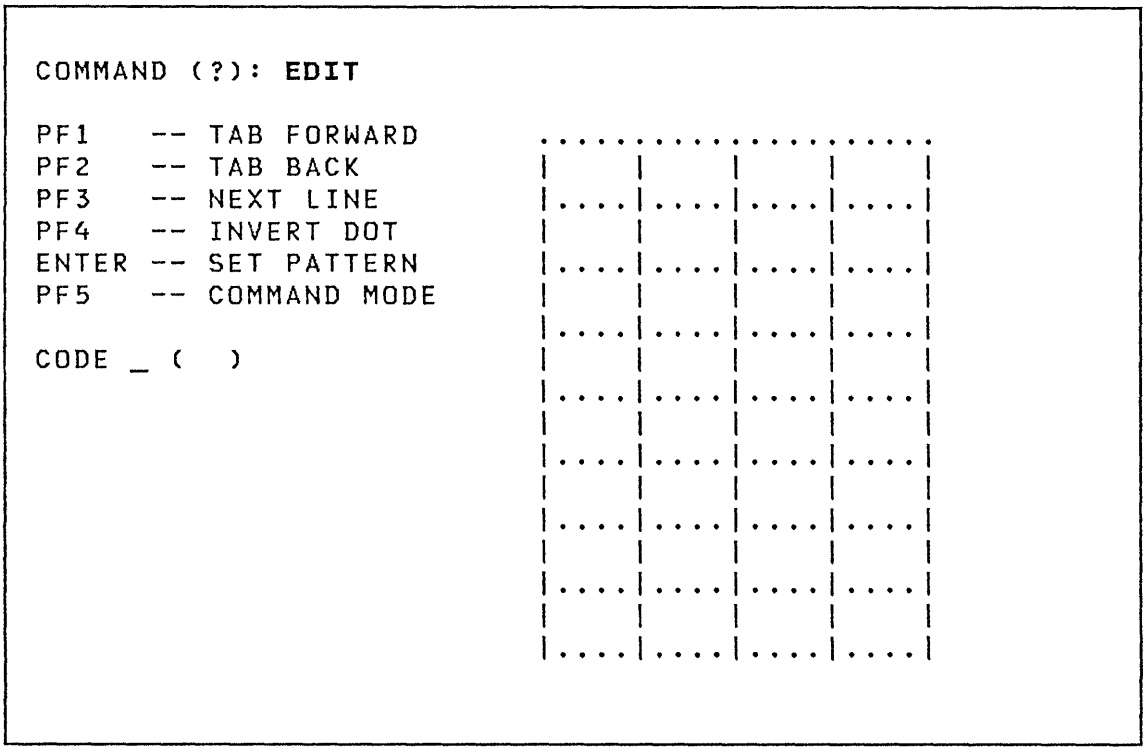
SAVE Examples

```
COMMAND (?): SAVE
```

```
COMMAND (?): SAVE 4974
```

Creating or Modifying a Character Image Table

When edit mode is first entered, a 4 x 8 grid is displayed in the center of the screen as follows:



For each grid row, the center of seven overlapping dot areas occur at the centers of each of the four squares and at each of the three interior grid lines.

The cursor is positioned just to the right of the CODE prompt on the bottom left of the screen. You can now enter a character in the present cursor position, or move the cursor to the right and enter an EBCDIC code between the parentheses. Once you have entered a character, $FONT fetches the bit pattern for this character, displays the character image in the image grid, and places the cursor in the top left-hand square of the 4 by 8 image grid. The EBCDIC code for the character entered is placed in the parentheses to the right of the CODE prompt. The character is also displayed below the character image grid (4978 only - not 4979).

You can then proceed to move the cursor about within the image grid by the PF key functions described on the screen. The PF keys function 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, and position it in the leftmost square of the new line

PF4 Inverts the dot pattern and extends the crossbars downward

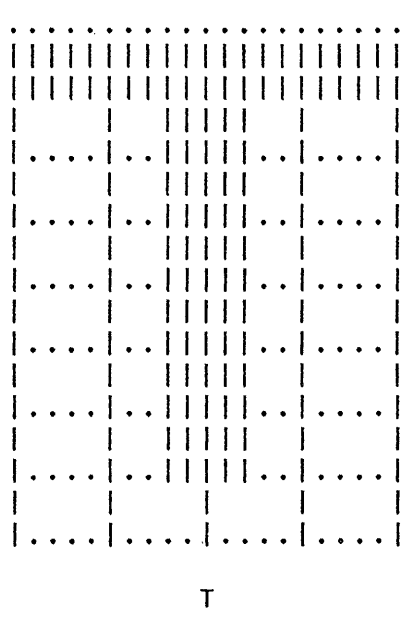
PF5 Returns you to command mode or cancels the current edit session. If the edit session is cancelled, the modified image will not replace the original image. To return to command mode once you have cancelled the edit session, press PF5 a second time.

The following shows a screen format after you select a character for display. In this example, the letter T was entered.

```
COMMAND (?): EDIT

PF1    -- TAB FORWARD
PF2    -- TAB BACK
PF3    -- NEXT LINE
PF4    -- INVERT DOT
ENTER  -- SET PATTERN
PF5    -- COMMAND MODE

CODE  T  (E3)
```



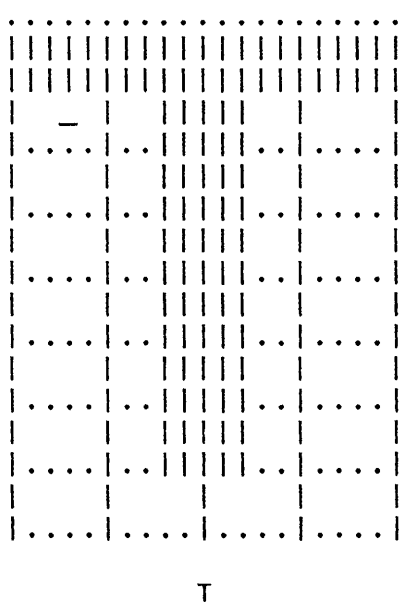
```
T
```

By moving the cursor about the image table with the PF keys, you can position the cursor on the screen for modification. In the following display, PF3 has placed the cursor in the position needed to begin modification of the T. The T will be altered by extending the ends of the top crossbar downwards. The cursor is now in a position to do that.

```
COMMAND (?): EDIT

PF1    -- TAB FORWARD
PF2    -- TAB BACK
PF3    -- NEXT LINE
PF4    -- INVERT DOT
ENTER  -- SET PATTERN
PF5    -- COMMAND MODE

CODE  T  (E3)
```



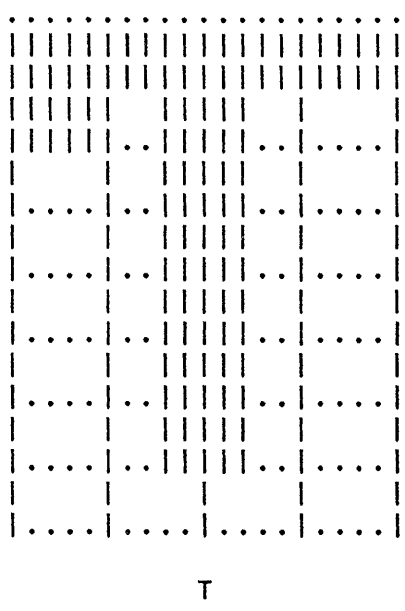
T

By pressing the PF4 key (invert dot), the dot pattern in the first square of the second line is inverted, which extends the left end of the crossbars downward. The image will appear as follows:

```
COMMAND (?): EDIT

PF1    -- TAB FORWARD
PF2    -- TAB BACK
PF3    -- NEXT LINE
PF4    -- INVERT DOT
ENTER  -- SET PATTERN
PF5    -- COMMAND MODE

CODE  T  (E3)
```



T

As modifications are made within the grid, they are reflected in the actual size character below the grid. You continue using the cursor to move about the grid and modify the image as desired.

Once you have completed modifying the image, press ENTER. The CODE prompt is changed to SET and the cursor is positioned just to the right of the SET prompt. At this point, you can either "set" the character just composed into the image table or cancel the changes made to the image. To "set" the character just composed into the image table, press ENTER and the modified T will replace the original T. To cancel the current edit session, press PF5. The SET prompt will be changed back to the CODE prompt and you can enter the character you wish to modify. If you press PF5 a second time, you are returned to command mode.

You also have the option of associating the modified T with a different key or EBCDIC code. If, for example, you type A on top of the T next to the SET prompt and then press ENTER, the modified T will replace the character A. If you move the cursor to the right and overtype the E3 within the parentheses with the EBCDIC code 0 (F0), the modified T image would replace that for 0.

By pressing the ENTER key without altering the character or EBCDIC code, the character is set, the CODE prompt is again displayed, and $FONT is ready for another character or EBCDIC code to be entered.

To end EDIT mode, press PF5 and reenter command mode. If you want to be able to load the modified image store to a 4978 or 4974 at a future time, the image store must be stored on disk or diskette. Use the SAVE command to store the modified image in the data set specified when $FONT was invoked.

Option 5 - SUBMIT

The SUBMIT option 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 the keyword DIRECT is entered 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: The DIRECT option is only to be used in systems with a HASP or JES2 interface.

Option 6 - LIST

The LIST option prints the entire contents of the work data set on the hardcopy device assigned to the terminal. (The listing can be terminated at any time by pressing the ATTN key and typing CA.)

Option 7 - MERGE

The MERGE option merges all, or part, of a source data set into the current edit work data set. You are prompted for the names of the Series/1 source data set and volume. If the specified data set is found, you are then prompted for the first and last line numbers of the data to be copied. You are also prompted for the line number of the current work data set after which the data is to be added. For example:

```
MERGE DATA FROM (NAME,VOLUME): SRCEDATA
                               LINES- 1ST LAST     10 430
                     ADD TO TARGET AFTER LINE #: 1050
```

If the entire source data set is to be merged, an asterisk is entered instead of the source data line specification, as follows:

```
MERGE DATA FROM (NAME,VOLUME): SRCEDATA
                               LINES- 1ST LAST     *
                     ADD TO TARGET AFTER LINE #: 490
```

The specification of an asterisk should only be used for the source data set. (If the format of the line number specification is incorrect, an error message is displayed and you are prompted for the data again.) 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.

```
MERGE DATA FROM (NAME,VOLUME): SRCEDATA
                               LINES- 1ST LAST     10 430
                     ADD TO TARGET AFTER LINE #: 1050
```

Caution: Once the merge has started, it must be allowed to complete normally or unpredictable results may occur. Series/1 source data sets are defined to consist of 80 character lines which are numbered in columns 73-80.

This page intentionally left blank.

Option 8 - END

The END option terminates $FSEDIT.

Option 9 - HELP

The HELP option displays tutorial text on the use of $FSEDIT.

Primary Commands

Primary commands are entered on line 2 of the display in the Command Input Field. All primary commands can be entered while in edit mode. In browse mode, three primary commands are recognized by $FSEDIT: LOCATE, FIND, MENU.

Most of the primary commands can be entered in abbreviated format. Only the first character is required. Minimum free form format is indicated with each command enclosed in ().

The function of each of the primary commands is described on the following pages.

$IAMUT1 - BUILD AND MAINTAIN INDEXED DATA SET

$IAMUT1 helps you manage your indexed data sets. The $IAMUT1 utility is shipped with an input/output buffer of 512 decimal bytes. This allows you to define an indexed data set with a maximum block size of 512 bytes, and to load, unload, and reorganize indexed data sets with a maximum record size of 512 bytes. If you want to change the maximum record size, refer to the Program Directory for use with Version 1, Modification level 2 of program 5719-AM3, section J, Programming Considerations

$IAMUT1 can be invoked using the $L command, $JOBUTIL, or the Session Manager.

$IAMUT1 Commands

The commands available under $IAMUT1 are listed below. To display this list at your terminal, enter a question mark in response to the prompting message COMMAND (?):.

```
ENTER COMMAND (?): ?

CR - INVOKE $DISKUT1
EC - SET/RESET ECHO MODE
EN - END THE PROGRAM

SE - SET DEFINE PARAMETERS
DF _ DEFINE AN INDEXED FILE
DI _ DISPLAY CURRENT SE PARAMETERS
RE _ RESET CURRENT VALUES FOR DEFINE

LO - LOAD INDEXED FILE FROM SEQUENTIAL FILE
RO - REORGANIZE INDEXED FILE
UN - UNLOAD INDEXED FILE TO SEQUENTIAL FILE

ENTER COMMAND (?):
```

After the commands are displayed, you are again prompted with COMMAND (?):. You respond with the command for of your choice (for example, EC).

CR - Create Data Set

CR allocates space for your data set in a volume by internally invoking the $DISKUT1 utility. The SE command should have been used to determine the number of records to allocate. When CR is entered on a terminal, the $DISKUT1 utility is loaded. You can then use the AL command of $DISKUT1 to allocate a data set; any other $DISKUT1 function can also be performed. Communication to the $DISKUT1 utility continues until the END command (EN) is entered, at which time communication to $IAMUT1 is restored. For information on the $DISKUT1 utility, refer to "$DISKUT1 - Allocate/Delete; List Directory Data" on page 135.

Note: Echo mode is not active during use of $DISKUT1.

The following example shows a use of the CR command:

```
ENTER COMMAND (?): CR
$DISKUT1 ACTIVE
USING VOLUME EDX002

COMMAND (?): AL SAMPLE1
HOW MANY RECORDS? 72
DEFAULT TYPE = DATA - OK? Y
SAMPLE1 CREATED

ENTER COMMAND (?):
```

DF - Define Indexed Data Set

DF command defines an indexed data set using an existing data set and the information you specify. When DF is entered, you are prompted for the immediate write back option and the names of the data set and volume to be formatted. Size calculations are made and the data set is formatted. The size calculation information is returned to you at your terminal on completion of the DF command. Before entering DF, you must use the SE command to set up parameters that determine the size and format of the indexed data set. The data set must have been allocated previously (the CR command can be used to allocate it).

DF allows you to select the immediate write back option. With this option, each modification to the indexed data set records that is the result of your request to update a data record is written to the data set immediately, thus contributing to the integrity of the data set. If you enter N to the immediate write back prompt, modifications are held in the main storage buffers for a period of time before being written back to the indexed data set. In most cases, not using the immediate write back option often results in fewer I/O operations and in better performance.

The following example shows a use of the DF command:

```
ENTER COMMAND (?): DF
DO YOU WANT IMMEDIATE WRITE-BACK? N
ENTER DATASET (NAME,VOLUME): SAMPLE1,EDX002
```

After entering the above information, the following is displayed:

$IAMUT1

```
TOTAL LOGICAL RECORDS/DATA BLOCK:          3
FULL RECORDS/DATA BLOCK:                   2
INITIAL ALLOCATED DATA BLOCKS:            50
INDEX ENTRY SIZE:                         32
TOTAL ENTRIES/INDEX BLOCK:                 7
FREE ENTRIES/PIXB:                         1
RESERVE ENTRIES/PIXB(BLOCKS):              0
FULL ENTRIES/PIXB:                         6
RESERVE ENTRIES/SIXB:                      0
FULL ENTRIES/SIXB                          7
DELETE THRESHOLD ENTRIES:                  7
FREE POOL SIZE IN BLOCKS:                  0
# OF INDEX BLOCKS AT LEVEL   1:            9
# OF INDEX BLOCKS AT LEVEL   2:            2
# OF INDEX BLOCKS AT LEVEL   3:            1

DATA SET SIZE IN EDX RECORDS:             73
INDEXED ACCESS METHOD RETURN CODE:        -1
SYSTEM RETURN CODE:                       -1
ENTER COMMAND (?):
```

DI - Display Parameter Values

DI displays the current parameter values entered via the SE command. The parameter values can be used to format a data set via the DF command or they can be modified by reusing the SE command.

The following example shows a use of the DI command.

```
ENTER COMMAND (?): DI
CURRENT VALUES FOR SE COMMAND ARE:
BASEREC           100
BLKSIZE           256
RECSIZE            80
KEYSIZE            28
KEYPOS              1
FREEREC             1
FREEBLK            10
RSVBLK           NULL
RSVIX               0
FPOOL            NULL
DELTHR           NULL
DYN              NULL
ENTER COMMAND (?):
```

EC - Control Echo Mode

EC allows you to enter or leave echo mode. When in echo mode, all $IAMUT1 input and output is logged on the $SYSPRTR device. This allows you to save information about the data sets you maintain using $IAMUT1. When in echo mode, all input and output is logged until either the current utility session is ended or echo mode is reset by use of the EC command. Echo mode is off when $IAMUT1 is loaded.

Note: Input and output from $DISKUT1 (CR command) is not logged.

The following examples show the commands to set and reset echo mode:

```
ENTER COMMAND (?): EC
DO YOU WANT ECHO MODE? (Y/N) Y    (Set echo mode)
ENTER COMMAND (?): EC
DO YOU WANT ECHO MODE? (Y/N) N    (Reset echo mode)
ENTER COMMAND (?):
```

RE - Reset Parameters

RE resets the parameters set up by the SE command to their default values.

The following example shows a use of the RE command:

```
ENTER COMMAND (?): RE
ENTER COMMAND (?):
```

$IAMUT1

SE - Set Parameters

SE prompts you for parameters that determine the structure and size of the indexed data set. The parameter values entered are saved by $IAMUT1. This allows you to reuse the SE command to change one or more parameters without having to reenter all of them. The current values can be displayed by the DI command.

Note: The values are retained only as long as $IAMUT1 remains loaded.

Size calculations are performed using the parameter values you have specified. The results are returned to you at your terminal. The SE command can be followed by the CR command to create a data set of the size specified on the SE command. After the data set has been allocated, or if a data set of the correct size already exists, the DF command can be used to format it.

The following list shows the default values for parameters on the SE command (all values are decimal):

BASEREC	NULL
BLKSIZE	0
RECSIZE	0
KEYSIZE	0
KEYPOS	1
FREEREC	0
FREEBLK	0
RSVBLK	NULL
RSVIX	0
FPOOL	NULL
DELTHR	NULL
DYN	NULL

If the default value is acceptable, enter a null line for the parameter when prompted for it. If you wish to change the value for any parameter, enter the new value in response to the prompting message. The new value becomes the new default value for the current $IAMUT1 session. The only parameters for which a null can be specified are BASEREC, FREEREC, FREEBLK, RSVBLK, RSVIX, FPOOL, DELTHR, and DYN. To specify a null parameter after the original default has been modified, enter an ampersand (&) in response to the prompting message. For an explanation of the SE command parameters, refer to "Determining Data Set Size and Format" on page 247.

The following example shows a use of the SE command in establishing the size and structure of an indexed data set.

$IAMUT1

```
ENTER COMMAND (?): SE
PARAMETER     DEFAULT NEW VALUE
BASEREC           NULL :100
BLKSIZE              0 :256
RECSIZE              0 :80
KEYSIZE              0 :28
KEYPOS               1 :1
FREEREC              0 :1
FREEBLK              0 :10
RSVBLK            NULL :
RSVIX                0 :
FPOOL             NULL :
DELTHR            NULL :
DYN               NULL :
```

After the above information has been entered, the following is displayed showing the size and structure of the defined indexed data set.

```
TOTAL LOGICAL RECORDS/DATA BLOCK:          3
FULL RECORDS/DATA BLOCK:                   2
INITIAL ALLOCATED DATA BLOCKS:            50
INDEX ENTRY SIZE:                         32
TOTAL ENTRIES/INDEX BLOCK:                 7
FREE ENTRIES/PIXB:                         1
RESERVE ENTRIES/PIXB(BLOCKS):              0
FULL ENTRIES/PIXB:                         6
RESERVE ENTRIES/SIXB:                      0
FULL ENTRIES/SIXB                          7
DELETE THRESHOLD ENTRIES:                  7
FREE POOL SIZE IN BLOCKS:                  0
# OF INDEX BLOCKS AT LEVEL   1:            9
# OF INDEX BLOCKS AT LEVEL   2:            2
# OF INDEX BLOCKS AT LEVEL   3:            1

DATA SET SIZE IN EDX RECORDS:             73
INDEXED ACCESS METHOD RETURN CODE:        -1
SYSTEM RETURN CODE:                       -1
ENTER COMMAND (?):
```

$IAMUT1

UN - Unload Indexed Data Set

UN unloads an indexed data set to a sequential data set. The record lengths of the two data sets need not be the same. Unloaded records are truncated or padded with zeros if the records lengths of the two data sets differ. Refer to the LO command.

Records are placed in the sequential data set in ascending key sequence as indicated by the indexed data set. Unloaded records are not blocked. They can span two or more 256 byte records. If the indexed data set contains more records than are allocated in the sequential data set, you are given the option to continue unloading to another sequential data set. If you choose to continue unloading, you are prompted for the name of the data set and volume to use to continue the unload operation. The unload operation continues, putting the records read from the indexed data set into the new sequential data set.

Note: The record length of subsequent output sequential data sets is assumed to be the same as the initial output sequential data set.

If the end of the output data set is reached and you choose not to continue, the unload operation ends.

Caution: Do not specify the same data set for input and output.

The following is an example of the commands and responses of an UN command:

```
ENTER COMMAND (?): UN
UNLOAD ACTIVE
ENTER INPUT DATASET (NAME,VOLUME): SAMPLE1,EDX002
ENTER OUTPUT DATASET (NAME,VOLUME): SAMPLE2,EDX002
OUTPUT RECORD ASSUMED TO BE 80 BYTES. OK?: N
ENTER RECORD SIZE: 256
UNLOAD IN PROCESS
END OF INPUT D/S
   100 RECORDS UNLOADED
UNLOAD SUCCESSFUL
ENTER COMMAND (?):
```

Building an Indexed Data Set

The SE and DF commands allow you to specify the size and format of your indexed data set and to do the actual data set formatting. Use the SE command to enter those values that determine the size of the indexed data set. Use the DF command to actually format the data set using the values previously specified on the SE command.

If the data set is too small to support the specified format, the amount of space required is returned to you. Knowing the available space and using the SE command, you can vary the SE parameters to design a data set of proper size.

Determining Data Set Size and Format

The data set design is determined by these SE command parameters:

BASEREC The estimated number of records to be loaded into the data set in ascending key sequence. These records can be loaded by $IAMUT1 or by a PUT request after either a LOAD or PROCESS request.

BASEREC should never be zero. If the DYN parameter is specified, the BASEREC parameter need not be specified, and defaults to 1. If the DYN parameter is not specified, the BASEREC parameter must be specified.

BLKSIZE The length, in bytes, of blocks in the data set. It must be a multiple of 256.

RECSIZE The length, in bytes, of records in the data set. Record length must not exceed block length minus 16.

KEYSIZE The length (1 to 254 bytes) of the key to be used for this data set.

KEYPOS The position, in bytes, of the key within the record. The first byte of the record is position 1.

FREEREC The number of free records to be reserved in each block. It should be less than the number of records per block (block size minus 16, divided by record size). It defaults to zero.

$IAMUT1

FREEBLK The percentage (0-99) of each cluster to reserve for free blocks. The percentage calculation result is rounded up so that at least one free block results. The calculation is adjusted to ensure that there is at least one allocated block in the cluster; that is, there cannot be 100% free blocks. This value defaults to zero.

RSVBLK The percentage of the entries in each lowest level index block to reserve for cluster expansion. These reserved entries are used to point to new data blocks as they are taken from the free pool to expand the cluster. The result of the calculation is rounded up so that any non-zero specification indicates at least one reserved index entry. The calculation is adjusted to ensure that there is at least one allocated block in the cluster. Enter a null character (*) for this prompt if you do not want initial reserved blocks and do not want the indexed access method to create reserved blocks as records are deleted and blocks become empty. Specify a value of zero for this prompt if you do not want initial reserved blocks but you do want the indexed access method to create reserved blocks as records are deleted and blocks become empty (See the DELTHR prompt). Note that the sum of the FREEBLK and RSVBLK prompts should be less than 100. This value defaults to null if the DYN parameter is not specified. If the DYN parameter is specified, this value defaults to zero.

RSVIX The percentage (0-99) of the entries in each second level index block to reserve for use in case of cluster splits. A cluster split is required when a cluster expands to its maximum potential size (as defined by the RSVBLK prompt) and another data set is inserted into the cluster. Each cluster split uses one reserved entry of the second level index block. The result of this calculation is rounded up so that any non-zero specification indicates at least one reserved index entry. The calculation is adjusted so that there is at least one unreserved entry in each second level index block. This value defaults to zero.

FPOOL The percentage (0-100) of the maximum possible free pool to allocate. The RSVBLK and RSVIX prompts result in a data set structure capable of drawing on the free pool for expansion. If insertion activity is evenly distributed throughout the data set, every reserve entry of every index block can be used. The number of blocks drawn from the free pool to support

this highly unlikely condition is the maximum free pool size needed for the data set. In more realistic cases, insertion activity is not evenly distributed throughout the data set, so fewer free blocks are needed. The percentage specified here represents the evenness of the distribution of inserted records. Specify a large number (90, for example) if you expect insertions to be evenly distributed. Specify a small number (20, for example) if insertions are anticipated to be concentrated in specific key ranges. If null is specified for this prompt, a free pool is not created for this indexed data set. If zero is specified, an empty free pool is created. Blocks can then be added to the free pool as records are deleted and blocks become empty (see the DELTHR prompt explanation). If you do not specify a null for this prompt, the RSVBLK must not be null and/or the RSVIX must be non-zero or an error is returned. Conversely, if the RSVBLK and/or RSVIX is non-zero, FPOOL must not be null or an error is returned.

DELTHR The percentage (0-99) of blocks to retain in the cluster as records are deleted and blocks made available. This is known as the delete threshold. When a block becomes empty, it is first determined if the block should be given up to the free pool by checking the response to this prompt. If the block is not given up to the free pool, it is retained in the cluster, either as a free block or as an active empty block. The result of this calculation is rounded up so that any non-zero specification indicates at least one block. If the DELTHR parameter is specified as null and the DYN parameter is not specified, this value defaults to the number of allocated blocks in the cluster plus one half of the value calculated by the FREEBLK prompt. If the DELTHR parameter is specified as null and a value is specified for the DYN parameter, this value defaults to zero. Specifiy null unless data set usage indicates that tuning is required.

DYN The number of blocks to be assigned to (or added to) the free pool. When this parameter is used with other free pool parameters, the free pool size is calculated as specified by the FPOOL parameter, and the value specified for DYN is added to the number of blocks for the free pool. DYN can be specified without the other parameters and the free pool will be the number of blocks specified for DYN.

If a value is specified for the DYN parameter, other parameters assume the following defaults if specified as null:

BASEREC = 1

FREEREC = 0

FREEBLK = 0

RSVBLK = 0

RSVIX = 0

DELTHR = 0

FPOOL = 0

The define (DF) command sets the size of the data set. Therefore, the BASEREC, FREEREC, FREEBLK, RSVBLK, RSVIX, FPOOL, and DYN parameters should be large enough to accommodate the maximum number of records planned for the data set.

To calculate the size of the data set for a given combination of parameters, use the SE command.

This page intentionally left blank.

$IMAGE

$IMAGE - DEFINE 4978/4979 FORMATTED SCREEN IMAGE

$IMAGE defines formatted screen images for the 4978/4979 display terminals, or for any terminal whose support includes the static screen functions. The images (formatted screens), which are defined in terms of protected and non-protected alphanumeric fields, can be saved in disk or diskette data sets, to be retrieved later by application programs for display, or by this utility for modification.

You must allocate the data set where the image will be stored before using $IMAGE. Most logical screens require a data set of two records. There are two modes of interaction with the $IMAGE program: command mode and edit mode. For retrieval information, refer to screen formatting in the System Guide.

You can find an example using $IMAGE to format a static screen in the System Guide. 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.

Note: A 2K-byte buffer is used withing $IMAGE to manipulate the screen being created. If the screen is greater than 2K bytes, unpredicatable results may occur.

$IMAGE Commands

$IMAGE is in command mode when loaded. When $IMAGE is in command mode, you specify the function to be performed by entering an alphabetic code followed by a parameter list. You are prompted for each item that is not specified in advance and does not have a default value.

The commands available under $IMAGE are listed below. To display this list at your terminal, enter a question mark in response to the prompting message COMMAND (?):.

AL - Allocate Data Set

AL identifies a data set to be allocated. It returns a $DISKUT3 return code that can be used by the JUMP command.

Note: If an AL statement is coded between a PROGRAM and EXEC statement, the results may be unpredictable.

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 U indicates undefined

AL Example

```
AL        TEMPDS,EDX003 25 D
```

$JOBUTIL

DE - Delete Data Set

DE identifies a data set to be deleted.

Note: If a DE statement is coded between a PROGRAM and EXEC statement, the results may be unpredictable.

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

DE Example

```
DE          TEMPDS,EDX003
```

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 all of the subsequent diskettes mounted must be placed 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 00 must be in slot number 1 of the first magazine. You can use either or both of the diskette magazines, A and B.

Data Set Specification

If $MOVEVOL is invoked with the $L command, you are prompted to enter the names of the data sets and volumes to be used.

Figure 21 shows the parameter menu displayed when $MOVEVOL is invoked using the session manager. Enter the requested information and press ENTER.

```
$SMM0308: SESSION MANAGER $MOVEVOL PARAMETER INPUT MENU-
ENTER/SELECT PARAMETERS:              DEPRESS PF3 TO RETURN

    DISK       ($$EDXLIB,VOLUME) ==>

    DISKETTE   (NAME,VOLUME)   ==>
```

Figure 21. $MOVEVOL parameter input menu

$MOVEVOL

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 00 (for example, SAVE00).

      2) Create a directory of at least 2 records.

      3) If the diskette is to 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, SAVE00) 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.

   c. Use $COPYUT1 to:

      1) Copy other data sets onto the control data set. For example, you may required $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 four-character prefix as the control diskette and a two-character suffix indicating the sequence number, for example,

$TAPEUT1 - TAPE MANAGEMENT (only available in Version 2, 5719-UT4)

$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.

$TAPEUT1 is invoked with the $L command or primary option 3 of the session manager. Once invoked and prior to accepting commands, $TAPEUT1 displays the following information about the tapes defined to the system:

- TAPEID
- Density selection (800, 1600, DUAL)
- Label type (SL, NL, BLP)
- Current density setting
- Online or offline
- Volume information (if an online SL tape)
- Device address

Example: Loading $TAPEUT1 and its automatic display of the system tapes.

```
> $L $TAPEUT1
$TAPEUT1     21P,11:11:33, LP= 0000
TAPE01 DUAL SL 1600 OFFLINE
   DEVICE ADDRESS = 004C
TAPE02 DUAL NL 1600 OFFLINE
   DEVICE ADDRESS = 004D

COMMAND (?):
```

Note: Error logging of tape errors is available. Refer to the System Guide.

$TAPEUT1 Commands

The commands available under $TAPEUT1 are listed below. To display this list at your terminal, enter a question mark in reply to the prompting message COMMAND (?):.

```
COMMAND (?): ?

CD - COPY DATASET
CT - CHANGE TAPE ATTRIBUTES
DP - DUMP TAPE
EN - END $TAPEUT1
EX - EXERCISE TAPE
IT - INITIALIZE TAPE
MT - MOVE TAPE
RT - RESTORE DISK/VOL FROM TAPE
ST - SAVE A DISK/VOL ON TAPE
TA - ALLOCATE TAPE DATASET

COMMAND (?):
```

After the commands are displayed, you are again prompted with COMMAND (?):. You respond with the command of your choice (for example, CD).

$TERMUT2

$TERMUT2 is used to:

- Restore the image buffer of a 4974 printer to the standard 64 character set.
- Assign a DEFINE key in a 4978 control store.
- Change the definition of one or more keys in a 4978 control store.
- Load a 4978 control store from a direct access data set or save a newly redefined 4978 control store into a direct access data set.
- Load a 4978 image store from a direct access data set or save a 4978 image store into a direct access data set. Refer to the description of the $FONT utility program for a description of image store definition.

You may wish to invoke these functions from a terminal other than the one you are using; therefore, you are requested to specify a terminal. If the selected terminal is not a 4974 or 4978, you are notified and the command is rejected.

4974 Support

Use $TERMUT2 to restore the image buffer of a 4974 printer to the standard 64 character set. The 4974 printer uses the Extended Binary Coded Decimal Interchange Code (EBCDIC), which includes 64 standard characters and five characters for international use. The standard key definition can be changed by using the TERMCTRL instruction within your application program or the $FONT utility and the redefined character set is stored in the image buffer of the 4974. For detailed information on the 4974 printer, refer to the Bibliography for the 4974 Printer manual.

4978 Support

Use $TERMUT2 to make special character string assignments on the 4978 keyboard. The functional characteristics of a 4978 keyboard are defined 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

Control data consists of scan code translation tables and a redefine table. Image data consists of a character image table.

4978 displays have a control store and an image store, which are loaded from disk or diskette. At IPL, the system automatically loads all 4978s with the control store data set $4978CS0 and the image store data set $4978IS0. These may be standard system-supplied data sets, or may be control/image store data sets that you created and renamed $4978CS0 or $4978IS0.

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.

For detailed information on the 4978 display station functions and the 4978 keyboards, refer to the Bibliography for 4978 Display Station manuals.

Data Set Names and Requirements

Two special names are reserved by the system and used during initial program load time:

```
$4978IS0  -  image store label

$4978CS0  -  control store label
```

These data sets are automatically searched for and loaded to defined 4978 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. The size of the data sets is as follows:

1. A 16-record (4096 bytes) data set for each control store

2. An 8-record (2048 bytes) data for each image store

Naming conventions for image store and control store data sets follow the conventions of the Event Driven Executive: the label can be up to eight characters.

$TERMUT2 Commands

The commands available under $TERMUT2 are listed below. To display this list at your terminal, enter a question mark in response to the prompting message COMMAND (?):.

```
COMMAND (?):  ?

AD - ASSIGN DEFINE KEY
C  - CHANGE KEY DEFINITION
EN - END PROGRAM
LC - LOAD CONTROL STORE
LI - LOAD IMAGE STORE
RE - RESTORE 4974 TO STD. 64 CHAR. SET
SC - SAVE CONTROL STORE
SI - SAVE IMAGE STORE

COMMAND (?):
```

After the commands are displayed, you are again prompted with COMMAND (?):. You respond with the command of your choice (for example, AD).

$TERMUT2

AD - Assign a Define Key

The AD command predefines a key that causes the 4978 attachment to enter define mode and enables you to assign a function or a series of functions to a specific key (for example, a PF 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 is being modified; * specifies the terminal that invoked $TERMUT2

AD Example

Assign a Define Key:

```
COMMAND (?):  AD
  ENTER SCAN CODE OF KEY TO BE ASSIGNED
    AS THE DEFINE KEY (HEX): 21
  ENTER TERMINAL NAME (CR OR * = THIS ONE): $SYSLOG
```

C - Change a Key Definition

The C command changes the function ID, local character, or interrupt code of a key to another definition on the terminal specified.

Syntax

```
C          terminal key id char interrupt

Required: ALL
Default:  none
```

Operands	Description
terminal	The name of the terminal whose control store is being modified; * specifies the terminal that invoked $TERMUT2
key	Defines the key (in hexadecimal) to be redefined
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

C 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 NEW FUNCTION\ID (HEX): 20
  ENTER NEW CHARACTER/FUNCTION CODE (HEX): 00
  ENTER NEW INTERRUPT CODE (HEX): 01
  ANOTHER KEY? N
```

EN - End Program

The EN command terminates the $TERMUT2 utility.

Syntax

```
EN

Required:  none
Default:   none
```

Operands Description

None

EN Example

Terminate $TERMUT2:

```
COMMAND (?): EN
$TERMUT2 ENDED
```

LC - Load a Control Store

The LC command loads 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; * specifies the terminal that invoked $TERMUT2

LC Example

Load a Control Store:

```
COMMAND (?): LC
  FROM DATA SET (NAME,VOLUME): $4978CS0,EDX002
  ENTER TERMINAL NAME (CR OR * = THIS ONE): *
```

LI - Load an Image Store

The LI command loads 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; * specifies the terminal that invoked $TERMUT2

LI Example

Load an Image Store:

```
COMMAND (?): LI
  FROM DATA SET (NAME,VOLUME): $4978ISO,EDX002
  ENTER TERMINAL NAME (CR OR * = THIS ONE): *
```

RE - Restore 4974 to Standard 64 Character Set

The RE command restores 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

RE Example

Restore 4974 to Standard 64 Character Set:

```
COMMAND (?): RE
ENTER TERMINAL NAME (CR OR * = THIS ONE): MPRINTER
```

SC - Save a Control Store

The SC command allows you 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 data set to be saved
volume	Define the name of volume where data set is to be saved
terminal	Defines the name of terminal issuing the save; * specifies the terminal that invoked $TERMUT2

SC Example

Save a Control Store::

```
COMMAND (?): SC
  SAVE DATA SET (NAME,VOLUME): $4978CS0,EDX002
  ENTER TERMINAL NAME (CR OR * = THIS ONE): $SYSLOG
```

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 data set is to be saved
terminal	Defines the name of the terminal issuing the save; * specifies the terminal that invoked $TERMUT2

SI Example

Save an Image Store:

```
COMMAND (?): SI
  SAVE DATA SET (NAME,VOLUME): $4978ISO,EDX002
  ENTER TERMINAL NAME (CR OR *=THIS ONE): $SYSLOG
```

Procedure for Defining an Interrupt Key

The 4978 hardware supports the DEFINE function, which allows keys to be defined with special character strings that have meaning to a particular application or job. In order to define a key with special characters, DEFINE mode must be entered. This is accomplished by pressing the DEFINE key on the 4978 keyboard.

Assuming a standard 4978 keyboard is installed, the $4978CS0 control store supports the keyboard in Figure 21.1. (The unshaded keys produce hardware interrupts.) As can be seen, a DEFINE key is not permanently designated. However, using the AD command of $TERMUT2, you may assign a key of your choice as the DEFINE key. In Figure 21.1, the key labeled **A** is being assigned as the DEFINE key and the key labeled **B** is the redefined interrupt key.

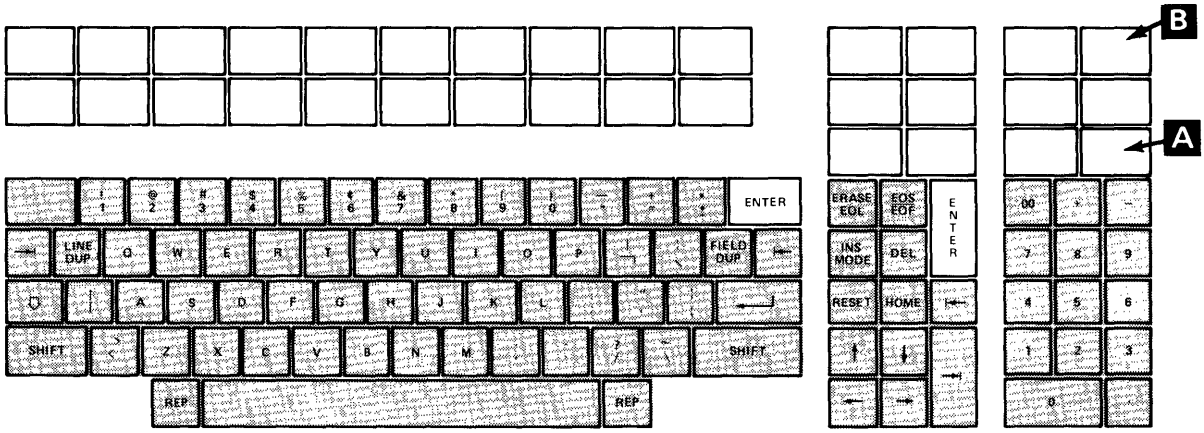


Figure 21.1 4978 Keyboard, RPQ D02056

Figure 21.2 and Figure 21.3 are taken from the General Information manual for the 4978 keyboard (RPQ D02056). Similar charts are in the General Information manuals for whichever keyboard you have installed.

In Figure 21.2, each key position is assigned a reference number. Figure 21.3 is a copy of the first of three pages documented in the General Information manual for this particular keyboard and lists the hex scan code, function ID code, local function code, and interrupt code that make up the control store information for each key. This page shows identifying numbers on the keys in Figure 21.2 corresponding to the key position numbers on the chart in Figure 21.3.

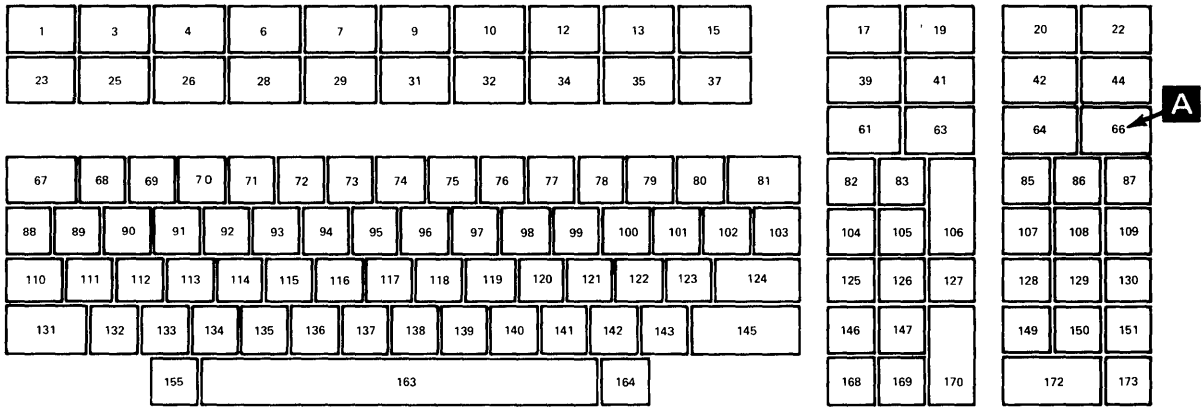


Figure 21.2 Keyboard Reference Assignments

Control Store Data

Downshift – Unshifted															*Upshift Shifted*														
Key position	*Scan code*	*Function ID code*	*Character/local function code*	*Interrupt code*	*Character image table Row* 0	1	2	3	4	5	6	7	*Keytop symbol*	*Key position*	*Scan code*	*Function ID code*	*Character/local function code*	*Interrupt code*	*Character image table Row* 0	1	2	3	4	5	6	7	*Keytop symbol*
1	01	20	00	01									Note 1	1	81	20	00	01									Note 1
3	02	20	00	02										3	82	20	00	02									
4	03	20	00	03										4	83	20	00	03									
6	04	20	00	04										6	84	20	00	04									
7	05	20	00	05										7	85	20	00	05									
9	06	20	00	06										9	86	20	00	06									
10	07	20	00	07										10	87	20	00	07									
12	08	20	00	0B										12	88	20	00	0B									
13	09	20	00	0C										13	89	20	00	0C									
15	0A	20	00	0D										15	8A	20	00	0D									
17	0B	20	00	0E										17	8B	20	00	0E									
19	0C	20	00	0F										19	8C	20	00	0F									
20	0D	20	00	10										20	8D	20	00	10									
22	0E	20	00	11										22	8E	20	00	11									
23	0F	20	00	12										23	8F	20	00	12									
25	10	20	00	13										25	90	20	00	13									
26	11	20	00	14										26	91	20	00	14									
28	12	20	00	15										28	92	20	00	15									
29	13	20	00	16										29	93	20	00	16									
31	14	20	00	17										31	94	20	00	17									
32	15	20	00	18										32	95	20	00	18									
34	16	20	00	19										34	96	20	00	19									
35	17	20	00	1A										35	97	20	00	1A									
37	18	20	00	1B										37	98	20	00	1B									
39	19	20	00	1C										39	99	20	00	1C									
41	1A	20	00	1D										41	9A	20	00	1D									
42	1B	20	00	1E										42	9B	20	00	1E									
44	1C	20	00	1F										44	9C	20	00	1F									
A 61	1D	20	00	20										61	9D	20	00	20									
63	1E	20	00	21										63	9E	20	00	21									
64	1F	20	00	22										64	9F	20	00	22									
66	20	20	00	23										66	A0	20	00	23									
67	21	70	00	00	00	00	00	00	00	00	00	00	(Blank)	67	A1	70	00	00	00	00	00	00	00	00	00	00	(Blank)
68	22	00	F1	00	02	06	02	02	02	02	07	00	1	68	A2	00	5A	00	07	30	30	02	02	00	02	00	!
69	23	00	F2	00	07	48	01	30	04	40	78	00	2	69	A3	00	7C	00	07	48	58	58	40	40	3C	00	@
70	24	00	F3	00	78	01	10	31	08	48	07	00	3	70	A4	00	7B	00	05	78	05	05	78	05	00	00	#
71	25	00	F4	00	28	28	0C	48	78	08	08	00	4	71	A5	00	5B	00	08	3C	50	07	28	71	40	00	$
72	26	00	F5	00	78	40	71	08	08	48	07	00	5	72	A6	00	6C	00	4C	45	10	02	20	0D	49	00	%
73	27	00	F6	00	02	20	04	47	48	48	07	00	6	73	A7	00	4A	00	10	3C	50	50	50	3C	10	00	¢
74	28	00	F7	00	78	01	10	02	20	04	40	00	7	74	A8	00	50	00	30	05	30	06	50	41	78	00	&
75	29	00	F8	00	30	05	30	05	48	48	07	00	8	75	A9	00	5C	00	00	05	30	78	30	05	00	00	*
76	2A	00	F9	00	07	48	48	0F	01	10	02	00	9	76	AA	00	4D	00	10	02	20	20	20	02	10	00	(
77	2B	00	F0	00	30	05	48	4A	48	05	30	00	0	77	AB	00	5D	00	20	02	10	10	10	02	20	00	)

Figure 21.3 Control Store Data

In Figure 21.1, assume you want to make the key at A the DEFINE key. In Figure 21.2, that key position has a reference of 66. When using the AD command, you are prompted for the scan code of the key to be assigned as the DEFINE key. On Figure 21.3, the scan code for key position 66 is hex 20. After the scan code and terminal have been entered, as follows:

```
COMMAND (?): AD
  ENTER SCAN CODE OF THE KEY TO BE ASSIGNED
    AS THE DEFINE KEY (HEX): 20
  ENTER TERMINAL NAME (CR OR * = THIS ONE): *
```

$TERMUT2 reloads the control store of the display, with key position 66 assigned as the DEFINE key.

You can now press the DEFINE key at A to enter DEFINE mode and screen indicator 2 on the display monitor will brighten to indicate DEFINE mode. The next key pressed after the DEFINE key is the key which will be redefined. In this example, the key to be redefined is interrupt key 22 (B). Now all key depressions, until the DEFINE key is again pressed, will be assigned to interrupt key 22.

You can now enter the character string $L $EDIT1N EDITWORK and then press one of the two ENTER keys. The screen will remain blank. You then press the DEFINE key again, ending the redefinition of interrupt key 22, and take the 4978 out of DEFINE mode.

The character string entered is a request to load the text editing utility program $EDIT1N, along with the name of a text edit work data set, EDITWORK.

Counting the pressing of the attention key required to get the > prompt, and the pressing the ENTER key following the load request, this line of text normally takes twenty-one keystrokes to enter the system. Now that interrupt key 22 has been redefined as this line of text, only three keystrokes are required:

- the attention key, resulting in the > prompt
- the interrupt key 22, which enters $L $EDIT1N EDITWORK
- the ENTER key, which was also part of the redefinition string.

For normal terminal usage, an active DEFINE key is not desirable. If it is pressed inadvertently, altering of the control store will result. The C command is used to change key position 66 back to its original control store configuration, using the chart in Figure 21.3 to supply the codes.

```
COMMAND (?): C
ENTER TERMINAL NAME (CR OR * = THIS ONE): *
ENTER SCAN CODE OF THE KEY TO BE REDEFINED (HEX): 20
ENTER FUNCTION ID (HEX): 20
ENTER CHARACTER/FUNCTION CODE (HEX): 00
ENTER INTERRUPT CODE (HEX): 23
ANOTHER KEY? N
```

At the conclusion of the C operation, the control store of your 4978 still has interrupt key 22 defined with the text editor load request character string, but with no DEFINE key designated. The SC command reads the control store, and stores it in a 16 record data set name 4978EDIT, which must be preallocated. Any time you desire a keyboard with interrupt key 22 designated as text editor load request, the LC command of $TERMUT2 can be used to load the control store from 4978EDIT.

```
COMMAND (?): SC
  SAVE DATA SET (NAME,VOLUME): 4978EDIT
  ENTER TERMINAL NAME (CR OR * = THIS ONE): *

COMMAND (?): EN

$TERMUT2 ENDED AT 01:27:44
```

Note: If a volume is not specified, the data set is saved on the IPL volume.

This page intentionally left blank.

$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, the terminal the message is to be send to , and if additional messages are to be sent. There are no commands, only prompting messages as follows:

```
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 LINE?
Yes or No.
If yes, you are prompted to enter another message line.

ANOTHER TERMINAL?
Yes or No.  If yes, select another terminal and
repeat the above process.

SEND MESSAGE LATER ('ATTN SEND')?
Yes or No.  If yes, the program remains loaded
and inactive.  Pressing the ATTN key and
entering 'SEND' starts the above process again.
```

Keep $TERMUT3 Active, Send a Message Later

```
      .
      .
      .
ANOTHER LINE ? N

ANOTHER TERMINAL ? N

SEND MESSAGE LATER ('ATTN SEND') ? Y

> SEND
ENTER TERMINAL NAME: TTY30
ENTER MESSAGE TO SEND
TTY30 - ARE YOU THERE

ANOTHER LINE ? N

ANOTHER TERMINAL ? N

SEND MESSAGE LATER ('ATTN SEND') ? N

$TERMUT3 ENDED AT 01:42:15
```

$TRAP

$TRAP - SAVE STORAGE ON ERROR CONDITION

$TRAP intercepts certain class interrupts and saves the contents of hardware registers and processor storage on a disk or diskette data set. This utility helps you to diagnose system or application program problems. The companion utility, $DUMP, prepares a formatted display of the data saved by $TRAP.

$TRAP must be executed before an expected failure so that the requested class interrupts can be intercepted when they occur. It can also be activated if a class interrupt such as a program check or machine check do not occur.

Two methods are provided to accomplished forced traps:

- ATTN: TRAPDUMP
- The console interrupt button

$TRAP does not affect the execution of any program or operator command unless it is activated by a class interrupt such as a program check or a machine check, or by the operator.

If the dump data set resides on a diskette and the diskette is mounted on a 4966, the diskette must be in slot 1.

Example

```
> $L $TRAP
                                                          Notes
 DUMPDS(NAME,VOLUME): DUMP,EDX003                          (1)
 $TRAP    15P,12:16:42,   LP8F00

 *-*  WARNING  *-*-*-*  WARNING  *-*-*-*  WARNING  *-*

 TO ACTIVATE TRAP USE     ATTN: TRAPON                     (2)
 TO DEACTIVATE TRAP USE   ATTN: TRAPOFF                    (2)
 TO FORCE TRAP USE        ATTN: TRAPDUMP                   (2)
 TO END $TRAP USE         ATTN: TRAPEND                    (3)
 DO NOT USE $C TO END $TRAP                                (3)
 IF TRAP OCCURS SYSTEM MUST BE IPLED                       (4)
 TRAP ON MACHINE CHECK? (Y/N) Y                            (5)
 TRAP ON PROGRAM CHECK? (Y/N) Y                            (6)
 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
 TRAP ON SOFT EXCEPTION? (Y/N) Y                           (7)
 INVALID FUNCTION? (Y/N) Y
 FLOATING POINT EXCEPTION? (Y/N) Y
 STACK EXCEPTION? (Y/N) Y
 TRAP ON CONSOLE INTERRUPT? (Y/N) Y                        (8)
 SAVE FLOATING POINT REGS? (Y/N) Y                         (9)

 TRAP SET OFF
 READY FOR ATTN: X---X                                     (11)

 > TRAPON                                                  (12)
 TRAP SET ON
 > TRAPOFF                                                 (12)
 TRAP SET OFF
```

Notes:

1. The $TRAP data set must be as large as the storage used: for a 64K-byte system, it must be at least 256 records; for a 128K-byte system at least 512 records.

2. Trap initiation is via the attention handler. TRAPON activates the trap facility but does not produce a storage dump. You must activate the trap facility (TRAPON) first; then specify TRAPDUMP to force a storage dump. If TRAPDUMP

is used, IPL to resume operations.

3. $TRAP should not be cancelled using $C. Use TRAPEND.

4. During trap processing, all I/O activities are halted. All configured devices are reset. IPL to resume operations.

5. A 'Y' response saves storage and hardware registers when any machine check (storage parity, CPU control or I/O error check) occurs.

6. A 'Y' response prompts you for the type of program check to trap. Any combination of the five types can be selected.

7. A 'Y' response prompts you for the type of soft exception to trap. Any combination can be selected.

8. It may be desirable to save storage even though no hardware detectable error has occurred. This option can be used to cause a trap when the console interrupt button is pressed. (Programmer console must be installed on Series/1)

9. Respond with 'Y' only if the Series/1 has the floating point hardware installed; otherwise no trap will take place.

10. Specify 'floating point exception' and 'save floating point regs' as 'n' if you do not have floating point hardware installed. If you specify 'Y' erroneously, $TRAP will attempt to trap the floating point registers and get an error which results in no dump to the data set.

11. $TRAP initializes to a TRAPOFF state. Any time after this message you can activate trap using the $TRAP attentiion commands. See notes 2-4.

12. These are the TRAPON and TRAPOFF attention commands.

$VERIFY

$VERIFY - INDEXED FILE VERIFICATION UTILITY

$VERIFY checks the validity of an indexed file and prints control block and free space information about the file on $SYSPRTR.

This $VERIFY description contains the following topics:

- $VERIFY Functions
- Invoking $VERIFY
- $VERIFY Example
- Messages Printed by $VERIFY
- $VERIFY Storage Requirements

$VERIFY Functions

With $VERIFY you can:

- Verify that all pointers in an indexed file are valid and that the records are in ascending sequence by key.
- Print a formatted File Control Block (FCB) listing, including the FCB Extension block. The FCB Extension block contains the original file definition parameters.

  Note: The FCB Extension block does not exist and file definition parameters are not saved in the FCB for indexed files defined prior to version 1.2 of the Indexed Access Method.

- Print a report showing the distribution of free space in your data set to help determine if reorganization is needed.

Invoking $VERIFY

$VERIFY can be invoked from either a terminal or a program coded in Event Driven Language. You must supply the same input in either case. If you invoke $VERIFY from a terminal, supply the input required in response to prompts. If you invoke $VERIFY from a program, supply the input required as parameters passed to the program.

$VERIFY Input

This section describes the input required to execute $VERIFY.

NAME,VOLUME Data set and volume names for the indexed file to be processed.

Option The type of processing you want $VERIFY to do. The three options are:

- Y - The FCB and the FCB Extension blocks are formatted and printed. The file is verified. A free space report is printed.
- N - The FCB and the FCB Extension blocks are formatted and printed. The file is verified. No free space report is printed.
- F - The FCB and the FCB Extension blocks are formatted and printed. The file is not verified. No free space report is printed.

Invoking $VERIFY From a Terminal

Load the $VERIFY program as follows:

```
$L $VERIFY
```

When $VERIFY begins execution, you are prompted for the parameters described previously. A complete example of a $VERIFY invocation from a terminal is shown under "$VERIFY Example" later in this chapter.

Invoking $VERIFY From a Program

$VERIFY

$VERIFY can be invoked by EDL programs with the LOAD instruction. The only required parameter is the address of a 16-byte area that contains:

	Hex Displacement	Decimal Length (Bytes)
Data set name	0	8
Volume name	8	6
Detail listing request (Y, N, or F left-justified)	E	2

The following example shows the use of $VERIFY to verify a data set named IAMFILE in the volume EDX002. A file verification and free space report are requested.

```
EXAMPLE  PROGRAM START,PGMS=($VERIFY)
START    EQU   *
          .
          .
          .
         LOAD  $VERIFY,PARMLIST
         WAIT  $VERIFY              WAIT FOR POST COMPLETE
          .
          .
          .
         PROGSTOP
PARMLIST EQU   *
DSNAME   DC    CL8'IAMFILE'         INDEXED DATA SET NAME
VOLUME   DC    CL6'EDX002'          VOLUME NAME
DETAIL   DC    CL2'Y'               PROCESSING OPTION
          .
          .
          .
         ENDPROG
         END
```

$VERIFY Example

This section presents the input and output for an example run of $VERIFY, along with descriptions of the material presented.

$VERIFY is invoked from the terminal as follows:

```
> $L $VERIFY
INDEXED ACCESS METHOD FILE VERIFICATION PROGRAM ACTIVE

(NAME,VOLUME): XMPL1,EDX002

DO YOU WANT DETAIL LISTING? (Y/N/F): Y

VERIFICATION COMPLETE,      0   ERROR(S) ENCOUNTERED

$VERIFY   ENDED
```

In this example, the first line loads and executes $VERIFY. The second line is printed by the program to indicate that execution has begun. In the third line, the program prompts for the data set name and volume of the indexed data set to be referenced by the program. The reply indicates the the data set is XMPL1, located on volume EDX002. In the fourth line, the program prompts for the amount of detail to be provided as output. The response of Y indicates that maximum detail is to be provided. As the program executes, it provides output to the printer, as shown in the example outputs that follow. Finally, messages are displayed to indicate the number of errors found and that the program has ended.

FCB Report

The first page of the example output from $VERIFY follows. This page is printed however the $VERIFY option is specified.

$VERIFY

```
VERIFY REPORT. NAME = XMPL1   ,  VOLUME = EDX002

FLAG1 :  DSHBL  TYPE
           1      1
KEY LENGTH=                                15
KEY DISPLACEMENT=                           0
BLOCK LENGTH=                             512
RECORD LENGTH=                            124
INDEX ENTRY LENGTH=                        20
RBN OF HIGH LEVEL IDX BLK IN USE=           2
RBN OF LAST DATA BLOCK IN USE=            108
RBN OF FIRST DATA BLOCK IN USE=             8
MAX # OF RECORDS PER DATA BLOCK=            4
MAX # OF ENTRIES PER INDEX BLOCK=          24
LOAD POINT VALUE FOR A DATA BLOCK=          1
LOAD POINT VALUE FOR AN INDEX BLOCK=        1
FLAG2 :  IMDWR
           1
VERSION NUMBER=                           1.2
DELETE THRESHOLD (RECORDS)=                 2
# OF AVAILABLE BLOCKS IN FREEPOOL          30
RBN OF 1ST FREE POOL BLOCK=               124
RBN OF HIGH PHYSICAL IX BLOCK=              2
LEVEL OF HIGH INDEX BLOCK IN USE=           3
```

This page of the example report is interpreted as follows:

The first line shows the data set name and volume.

FLAG1: These two lines show the significant bits of the first flag byte in the FCB. The first of the pair of lines are a heading, and the second the bit values (where 1 indicates the bit is set, 0 indicates it is not set). The headings are defined as follows:

DSHBL Data set has been loaded flag. This flag is set when any record has been successfully loaded into the data set in load mode.

TYPE This flag indicates whether the indexed data set was created with the Realtime Programming System Indexed Access Method PRPQ (bit=0) or either the Event Driven Executive or Realtime Programming System Indexed Access Method Program Product (bit=1).

KEY LENGTH Shows the length of the key in bytes.

KEY DISPLACEMENT Shows the byte displacement of the key from the start of the record.

BLOCK LENGTH Shows the byte length of blocks in the file.

RECORD LENGTH: Shows the byte length of records in the file.

INDEX ENTRY LENGTH: Shows the number of bytes in each index entry. This length should be the key length plus 4, rounded up to a multiple of two bytes.

RBN OF HIGH LEVEL IDX BLK IN USE: Shows which index block is to be used as the starting point when the index is to be searched.

RBN OF LAST DATA BLOCK IN USE: Points to the last logical data block in the data set.

RBN OF FIRST DATA BLOCK IN USE: Points to the first logical data block in the data set. It is used as the starting point when a sequential read operation is begun with no key specified.

MAX # OF RECORDS PER DATA BLOCK: Shows how many data records can be contained in a data block.

MAX # ENTRIES PER INDEX BLOCK: Shows how many index entries can be contained in an index block.

LOAD POINT VALUE FOR A DATA BLOCK: The number of records that can be placed in each data block while in load mode. This value is calculated at file definition time to provide the requested number of free records.

LOAD POINT VALUE FOR AN INDEX BLOCK: The number of data blocks in each cluster to be used while in load mode. This value is calculated at file definition time to provide the space requested by the RSVBLK, RSVIX and FREEBLK parameters.

FLAG2: Another byte of flags described by a pair of lines: a heading line followed by a data line. The heading has the following meaning:

IMDWR Immediate write back flag. If set (1), this flag indicates that the immediate write back option was specified when the indexed data set was defined.

$VERIFY

VERSION NUMBER: Shows the version number and modification level of the Indexed Access Method that was used to define the indexed data set.

DELETE THRESHOLD (RECORDS): Indicates the number of data blocks to retain in each cluster as records are deleted and blocks become empty. This value is calculated when the file is defined and is based on the DELTHR parameter.

 # OF AVAILABLE BLOCKS IN FREEPOOL: The number of available blocks in the free pool. This count is updated as blocks are taken from or returned to the free pool.

RBN OF 1ST FREE POOL BLOCK: Points to the last block which was put in the free pool (which is the next block to be taken from the free pool).

RBN OF HIGH PHYSICAL IX BLOCK: Points to the physical top of the index. In some cases (if the file has not been completely loaded), this RBN might not agree with the RBN OF HIGH LEVEL IDX BLK IN USE. If it does not agree, then the data set is structured with index blocks that are not yet needed because the file does not contain enough records.

LEVEL OF HIGH INDEX BLOCK IN USE: Indicates how many levels of the index are currently in use. This field was not used prior to version 1.2 of the Indexed Access Method.

FCB Extension Report

The second page of the example output from $VERIFY follows. This page is printed however the $VERIFY option is specified if the file was defined with version 1.2 of the Indexed Access Method or a later release.

This information is obtained from the FCB Extension block and shows the parameters that were specified when the file was defined. Some information (BLKSIZE, RECSIZE, KEYSIZE, KEYPOS) is duplicated on the FCB and FCB Extension report because it is contained in both control blocks. The values should correspond with each other. The word NULL for the value of a parameter indicates that the value was not specified when the file was defined (that is, it was allowed to default).

$VERIFY

```
VERIFY REPORT. NAME = XMPL1   ,  VOLUME = EDX002

DATASET WAS CREATED WITH THESE IDEF PARAMETERS:

BASEREC=              32
BLKSIZE=             512
RECSIZE=             124
KEYSIZE=              15
KEYPOS=                0
FREEREC=               3
FREEBLK=               3
RSVBLK=               91
RSVIX=                66
FPOOL=                 3
DELTHR=                7
DYN=                NULL
```

Note: The IDEF parameters are the file definition parameters that were specified on the SE command of the $IAMUT1 utility when the file was defined.

Free Space Report

A free space report of the example output from $VERIFY follows. The free space report is printed only if the $VERIFY option is specified as Y.

```
$VERIFY
```

```
VERIFY REPORT. NAME = XMPL1   ,  VOLUME = EDX002

          TOTAL    USED     UNUSED   RESERVE   FREE   AVAILABLE
RBN  LVL  ENTRIES  ENTRIES  ENTRIES  ENTRIES   BLOCKS RECORD SLOTS
---  ---  -------  -------  -------  -------   ------ ------------
 2    3     24        4        0       20        0         0

 3    2     24        8        0       16        0         0
 4    2     24        8        0       16        0         0
 5    2     24        8        0       16        0         0
 6    2     24       11        0       13        0         0

 7    1     24        1        0       22        1         8
10    1     24        1        0       22        1         8
13    1     24        1        0       22        1         8
16    1     24        1        0       22        1         8
19    1     24        1        0       22        1         8
22    1     24        1        0       22        1         8
25    1     24        1        0       22        1         8
28    1     24        1        0       22        1         8
31    1     24        1        0       22        1         8
34    1     24        1        0       22        1         8
37    1     24        1        0       22        1         8
40    1     24        1        0       22        1         8
43    1     24        1        0       22        1         8
46    1     24        1        0       22        1         8
49    1     24        1        0       22        1         8
52    1     24        1        0       22        1         8
55    1     24        1        0       22        1         8
58    1     24        1        0       22        1         8
61    1     24        1        0       22        1         8
64    1     24        1        0       22        1         8
67    1     24        1        0       22        1         8
70    1     24        1        0       22        1         8
73    1     24        1        0       22        1         8
76    1     24        1        0       22        1         8
79    1     24        1        0       22        1         8
82    1     24        1        0       22        1         8
85    1     24        1        0       22        1         8
88    1     24        1        0       22        1         8
91    1     24        1        0       22        1         8
94    1     24        1        0       22        1         8
97    1     24        1        0       22        1         8
25    1     24        6        0       18        0        12
26    1     24        5        0       19        0         7
47    1     24       13        0       11        0        21
VERIFICATION COMPLETE,      0    ERROR(S) ENCOUNTERED
```

$VERIFY

In this report, each printed line represents an index block. The columns have the following meanings:

RBN The relative block number within the indexed data set, based on the block size specified when the file was defined. The first block in the data set is relative block number zero.

LVL The level of the index block analyzed. Lowest level (PIXB) is 1, second level (SIXB) is 2, etc.

TOTAL ENTRIES The maximum number of index entries that can fit in an index block.

USED ENTRIES The number of entries used in this index block.

UNUSED ENTRIES The number of entries in the index block which are neither used nor reserved.

RESERVE ENTRIES The number of reserve entries in this index block. This number represents the number of new index blocks that can be obtained from the free pool for creation of new blocks, provided there are enough blocks remaining in the free pool.

FREE BLOCKS The number of free blocks associated with this index block.

AVAILABLE RECORD SLOTS The maximum number of records that can be inserted into this cluster without obtaining blocks from the free pool.

$VERIFY

$VERIFY Messages

As $VERIFY executes, any errors encountered result in an error message being written describing the type of error and where the error occurred.

File Error Messages

The following identify messages which indicate that the indexed data set contains errors:

BLOCKS IN FREEPOOL CHAIN DOES NOT MATCH FREE POOL COUNT IN FCB.

BLOCK OUT OF SEQUENCE. RBN _____.

HIGH KEY IN RBN _____ DOES NOT MATCH IX ENTRY IN RBN _____.

POINTERS IN HEADER OF HIGH INDX BLK ARE NOT ZERO.

RBN _____ CONTAINS INVALID UPWARD POINTER.

RBN _____ CONTAINS INVALID BACKWARD POINTER.

RBN _____ CONTAINS INVALID FORWARD POINTER.

RBN _____ IS IN FREEPOOL CHAIN, BUT IS NOT A VALID FREEPOOL BLOCK.

RBN _____ CONTAINS INVALID BACKWARD POINTER. TASK TERMINATED.

RECORD OUT OF SEQUENCE NEAR RBN _____.

If any of these messages are printed, the indexed file has at least one error.

Possible sources of the error include:

- The file is not an indexed data set
- Data in the file has been inadvertently destroyed
- The Indexed Access Method has a program error

$VERIFY Storage Requirements

$VERIFY has two types of storage requirements:

- Storage for the $VERIFY and $DISKUT3 code
- Storage for work space in verifying the file

The $VERIFY utility code requires approximately 9K bytes of storage. In addition to this, $DISKUT3 (which is invoked to open the indexed file) requires 4K bytes of dynamic storage. The space for $DISKUT3 must be available in the partition into which $VERIFY is loaded.

Working storage space is also needed (unless the $VERIFY option is specified as F). The size of this storage required varies, depending on the maximum number of blocks at the SIXB level and the block size of the file.

Using Default Working Storage Requirements

The default working storage specification is 8K bytes. For a file with a block size of 256, this default is sufficient to handle up to 1920 blocks at the SIXB level. The larger the block size of the file, the fewer the maximum number of SIXB's that can be processed.

The following formula can be used to calculate the maximum number of blocks at the SIXB level that $VERIFY can process, given the block size of the indexed data set:

```
NS = (8192 - (2 * BLKSIZE)) / 4
```

NS is the number of blocks at the SIXB level
BLKSIZE is the block size of the indexed data set

Modifying Working Storage Requirements

The default working storage allocation is intended to satisfy the requirements of most indexed files. It may be necessary or desirable to modify the amount of working storage space available to $VERIFY.

The following formula can be used to calculate the amount of working storage required to process a file with a given block size and number of blocks at the SIXB level.

$VERIFY

```
DS = (4 * NS) + (2 * BLKSIZE)
```

Where:
DS is the amount of dynamic storage required
NS is the number of blocks at the SIXB level
BLKSIZE is block size of the indexed data set

The number of SIXB's in a file can be determined by examining the free space report.

You can override the default working storage size at load time (if loaded by a program), or with the SS command of the $DISKUT2 utility.

Summary

$VERIFY requires 13K bytes plus a variable amount of working storage which defaults to 8K bytes. Increase the working storage size if $VERIFY runs out of space during execution. Decrease the working storage size if the number of SIXB's is significantly less than that supported by the default working storage allocation (1920 with a block size of 256) and space is at a premium.

This page intentionally left blank.

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:  TSTPRG
TSTPRG STORED

COMMAND (?):
```

End $UPDATE

```
COMMAND (?):  EN

$UPDATE ENDED AT 11:39:34
```

Invoking $UPDATE

Invoking $UPDATE Using $JOBUTIL

When $UPDATE is invoked as part of a batch job under the control of $JOBUTIL, certain restrictions apply to its operation. In this mode, the command is assumed to be RP. The Rename function is not supported; however, the Replace function is. Refer to the preceding examples for a description of Rename and Replace.

In batch mode, $UPDATE terminates its execution after performing one RP command. A completion code is set by $UPDATE depending upon the success or failure of the requested operation. This code can be tested by the JUMP command of $JOBUTIL. The $UPDATE completion codes are described in Chapter 6, Messages and Codes.

When $JOBUTIL is used to invoke $UPDATE, the information required by $UPDATE must be passed to it by means of 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

3. The name,volume of the data set to contain the output loadable program

4. An optional parameter YES if the output module is to replace an existing module of the same name,volume

The volume names of the data sets must be given unless they reside on the IPL 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.

An example of invoking $UPDATE via $JOBUTIL commands follows:

CHAPTER 6. MESSAGES AND CODES

This chapter describes messages and codes generated by the Event Driven Executive. The information is presented in two sections; the first contains messages information and the second contains the completion, return, and post codes.

Messages

MESSAGES

This section lists and describes messages generated while:

- Initializing your system
- Executing a utility
- Executing a program

The messages are documented in alphabetic order according to the message text. Messages that begin with a variable are documented at the end of the alphabetic listing under X.

Immediately following the alphabetic listing of messages, the program check error message, the system program check error message, and the processor status word (PSW) are documented.

Message Documentation

The documentation for each message includes the following.

Message Text: The message text is presented in boldface type, as it will be printed or displayed. Lowercase variable fields will be replaced with the correct data when the message is issued.

Issued By: The issued by text identifies the function that issued the message.

Explanation: The explanation identifies all the variable fields and describes any information supplied by the program. The conditions under which the message is issued are noted.

System Action: The system action describes the system reaction to the condition that caused the message to be issued.

Response: The response explains how you are to reply to the message or what, if any, corrective actions are required to help you resume.

Return Codes: The message text sometimes includes a return code, which is generated by the Event Driven Executive. The name of the function that generated the return code is specified in the "Issued By" portion of the message documentation.

To determine the error condition, get the name of the function involved and go to the Codes section of this chapter, which lists the return codes by function.

$DISKUT1 LOAD ERROR. RETURN CODE = xxx.

Issued By: $IAMUT1

Explanation: $IAMUT1 attempted to load the $DISKUT1 utility program and load failed.

System Action: Prompts for next command.

Response: Check the LOAD return code to find the cause of the problem and take the appropriate action.

$DISKUT3 LOAD ERROR

Explanation:

$VERIFY was unable to load $DISKUT3.

System Action:

An abend message follows with the return code from LOAD contained in it. $VERIFY terminates.

Response:

Examine the LOAD return code to determine the problem. Refer to the description of the LOAD return codes.

If the LOAD return code indicates insufficient storage, rerun $VERIFY in a partition with more storage available or rerun it when more space is available in the current partition.

Otherwise, correct the problem and retry.

Messages

$LOG - * INSUFFICIENT BUFFER FOR LOG RATE ***

Issued By: $LOG Utility

Explanation: The error has become full.

System Action: The $LOG utility is terminated.

Response: Increase the buffer size and restart the $LOG utility.

$RMU ERROR 1 - INSUFFICIENT BUFFER. SIZE: nnnn

Issued By: Remote Management Utility

Explanation: The size of the buffer defined for use by the utility is less than the 512 byte minimum. The default 1024 byte buffer size has been modified incorrectly.

System Action: The execution of the program terminates.

Response: Check the buffer size and respecify a valid size and restart the program.

$RMU ERROR 2 - COMMUNICATIONS OPEN FAILED, RETURN CODE: xxx

Issued By: Remote Management Utility

Explanation: The OPEN of the BSC communication line failed.

System Action: The execution of the program is terminated.

Response: Check the BSC return code to find the cause of the problem and take the appropriate action.

$RMU ERROR 3 - COMMUNICATIONS CLOSE FAILED, RETURN CODE: xxx

Issued By: Remote Management Utility

Explanation: The CLOSE of the BSC communication line failed.

System Action: The execution of the program terminates.

Response: Check the BSC return code to find the cause of the problem and take the appropriate action.

$RMU ERROR 4 - COMMUNICATIONS I/O ERROR.
I/O FUNCTION: aaaaaa
RETURN CODE: xxx

Issued By: Remote Management Utility

Explanation: A communication error has been detected by the utility.

System Action: This message appears whenever BSCAM encounters an I/O error. If the error is either a time-out (return code 10) or a sequence error (return code 12), the Remote Management Utility terminates the current function and waits for the next request from the host. Otherwise, the Remote Management Utility terminates after reporting the I/O error.

Response: The I/O function (aaaaaa) will indicate the type of request, and is one of the following:

READ INITIAL
READ CONTINUE
WRITE EOT
WRITE INITIAL
WRITE EOT (ABORT)
WRITE CONTINUE

Check the BSC return code to find the cause of the problem and take the appropriate action.

Messages

$RMU ERROR 5 - LOAD OVERLAY FAILED, RETURN CODE: xxx
OVERLAY NUMBER: mmmm

Issued By: Remote Management Utility

Explanation: The utility attempted to load an overlay program via a LOAD instruction, and the LOAD failed.

System Action: The execution of the program terminates.

Response: Check the LOAD return code to find the cause of the problem and take the appropriate action.

$RMU ERROR 6 - OVERLAY FUNCTION MISSING. FUNCTION: nnnn
OVERLAY NUMBER: mmmm

Issued By: Remote Management Utility

Explanation: The utility's function table defined a function as being contained within an overlay, but is was not.

System Action: The execution of the program is terminated.

Response: Check to see that the user written function is added properly to the function table. Correct the problem and restart the program.

$TRAP MUST BE IN PARTITION #1.
$TRAP TERMINATED

Issued By: $TRAP Utility

Explanation: $TRAP was loaded into a partition other than partition 1. It must be loaded into partition 1.

System Action: The $TRAP utility is terminated.

Response: Load $TRAP into partition 1 and restart the $TRAP utility.

$VERIFY ABENDED, RTCODE = ___

Explanation:

$VERIFY encountered an error such that it was unable to continue processing.

System Action:

$VERIFY terminates.

Response:

Check the previous message printed to determine the general problem area. Examine the return code in this message to determine the problem. Refer to the description of the return codes.

Correct the problem and retry.

BUFFER SMALLER THAN INPUT OR OUTPUT RECORD. INCREASE BUFFER SIZE BEFORE REATTEMPTING OPERATION

Issued By: $IAMUT1

Explanation: Insufficient program storage to contain an input or output record.

System Action: Command terminates.

Response: Increase the dynamic program size using the SS command of the $DISKUT2 utility. The buffer size must be large enough to contain the entire input and/or output record (whichever is larger).

Messages

CONNECT TO MTM?

Issued By: Multiple Terminal Manager

Explanation: This message is written to determine whether to connect the terminal to $MTM or some other program.

System Action: Accepts operator response.

Response: Enter 'Y' to connect the terminal to a Multiple Terminal Manager named $MTM. Enter 'N' to connect the terminal to a Multiple

CONNECTED TO MULTIPLE TERMINAL MANAGER

Issued By: Multiple Terminal Manager

Explanation: This message is written to a non-full-screen type terminal when it is connected to the Multiple Terminal Manager.

System Action: None.

Response: None needed.

DEVICE AT ADDRESS 0002 UNUSABLE

Issued By: Diskette Initialization

Explanation: A 4964 diskette device is marked unusable and the status message is printed next if I/O operations to the device fail.

System Action: The initialization is terminated.

Response: Check to see what problems might exist with the diskette. Correct the problem and restart the initialization.

DEVICE AT ADDRESS 0003 UNUSABLE

Issued By: Disk Initialization

Explanation: A 4962 or 4963 disk is marked unusable and the status message is printed next if I/O operations to the device fail.

System Action: The initialization is terminated.

Response: Check to see what problems might exist with the disk. Correct the problem and restart the initialization.

DEVICE AT ADDRESS 22 UNUSABLE

Issued By: Diskette Initialization

Explanation: A 4966 is searched in slot 1, slot 1 of magazine A and slot 1 of magazine B. If a diskette is found in any one of these slots, searching continues until an empty slot is found. If no diskettes in the 4966 diskette device can be accessed the device is marked unusable and the following status message is printed.

System Action: The initialization is terminated.

Response: Insert a new diskette in the 4966 diskette device and restart the initialization.

DEVICE TYPE INVALID

Issued By: Multiple Terminal Manager

Explanation: The device type specified for the TERMINAL file record listed immediately before this message is invalid or TYPE=3101 and no 3101 support was included.

System Action: The terminal is not connected.

Response: Correct the TERMINAL record or rebuild the Multiple Terminal Manager with the 3101 support. Stop and restart the manager.

Messages

DISCONNECTED FROM xxxxxxxx

Issued By: Multiple Terminal Manager

Explanation: The terminal is disconnected from the Multiple Terminal Manager named xxxxxxxx.

System Action: None.

Response: Reconnect the terminal when desired.

DISK ERROR DURING INITIALIZATION, RC=xxx

Issued By: Multiple Terminal Manager

Explanation: A disk error occurred while reading the SCRNS volume directory, the PRGRMS volume directory, the TERMINAL data set, or an error occurred while writing to the MTMSTORE data set.

System Action: Aborts initialization.

Response: Check the Multiple Terminal Manager return code to find the cause of the problem and take the appropriate action.

DISK READ ERROR

Issued By: Multiple Terminal Manager

Explanation: An internal Multiple Terminal Manager disk read error has occurred and results may be unpredictable.

System Action: Results are unpredictable.

Response: Determine the cause of the error and retry operation.

dsname ON dsvol DOES NOT CONTAIN $TRAP OUTPUT. $DUMP TERMINATED

Issued By: $DUMP Utility

Explanation: The data on the data set to be dumped is not the output of $TRAP.

System Action: The $DUMP utility is terminated.

Response: Enter the valid dsname and dsvol for that which is the output of $TRAP and restart the $DUMP utility.

dsname ON dsvol IS ONLY xxx RECORDS. MINIMUM SIZE IS yyy RECORDS.
$TRAP TERMINATED

Issued By: $TRAP Utility

Explanation: The data set assigned to $TRAP is not large enough to contain the amount of storage being saved.

System Action: The $TRAP utility is terminated.

Response: Increase the size of the data set assigned to $TRAP to a minimum of yyy records and restart the $TRAP utility.

Messages

DSOPEN ERROR. RETURN CODE = xxx. RETRY?

Issued By: $IAMUT1

Explanation: An attempt to open the specified data set failed due to one of the following reasons:

- Data set not found on specified volume
- Invalid IODA
- Invalid VOLSER
- Library not found
- I/O error occurred
- No VTOC
- DSNAME = $$

System Action: Allows user to respecify DSNAME,VOLUME if retry = Y. If retry = N, command terminates.

Response: Check the Indexed Access Method return code to find the cause of the problem and take the appropriate action.

DUMP RANGE INVALID VALID RANGE IS xxxx TO yyyy

Issued By: $DUMP Utility

Explanation: An invalid starting or ending storage address was entered during a partial storage dump.

System Action: The $DUMP utility is terminated.

Response: Enter a valid storage address in the range xxxx to yyyy and restart the $DUMP utility.

EDX SYSTEM INITIALIZATION COMPLETE

Issued By: System

Explanation: When the system is initialized, this message is printed.

System Action: None.

Response: None.

ENTER NAME OF MULTIPLE TERMINAL MANAGER PROGRAM

Issued By: Multiple Terminal Manager

Explanation: This message is printed by the $RECON utility. It results from an entry of 'N' to a prompting message asking if the terminal is to be connected to $MTM.

System Action: Accepts operator response.

Response: Enter the name of the program to which you want to connect the terminal.

ERROR ENCOUNTERED DURING CLOSE OF INDEXED ACCESS METHOD (ddddddd,vvvvvv), ERROR CODE=(xxx)

Issued By: Multiple Terminal Manager

Explanation: An error occurred during AUTOCLOSE of an Indexed Access Method data set.
ddddddd is the data set name.

vvvvvv is the volume name.

System Action: None.

Response: Check the Indexed Access Method return code to find the cause of the problem and take the appropriate action.

Messages

ERROR OCCURRED DURING DISK READ

Explanation:

$VERIFY attempted to read from the indexed data set and an unrecoverable disk read error occurred.

System Action:

An abend message follows with the return code from READ contained in it. $VERIFY terminates.

Response:

Examine the READ return code to determine the problem. Refer to the description of the READ return codes.

Correct the problem and retry.

ERROR ENCOUNTERED USING $DISKUT3 FOR OPEN

Explanation:

$VERIFY attempted to open the indexed data set and a $DISKUT3 error occurred.

System Action:

An abend message follows with the return code from $DISKUT3 contained in it. $VERIFY terminates.

Response:

Examine the $DISKUT3 return code to determine the problem. Refer to the description of the $DISKUT3 return codes.

Correct the problem and retry.

***** EVENT DRIVEN EXECUTIVE *****

Issued By: IPL Operation

Explanation: After $SYSLOG terminal initialization this message appears on the $SYSLOG terminal.

System Action: None.

Response: None.

I/O ERROR INITIALIZATION FIXED HEAD DEV, DISK RETURN CODE= xxx IN THE TWO RECORDS STARTING WITH RECORD xxx

Issued By: Disk Initialization

Explanation: An error was encountered during fixed head initialization starting with record xxx.

System Action: The initialization is terminated.

Response: Check the Disk return code to find the cause of the problem and take the appropriate action.

INITIALIZATION ERROR

Issued By: Multiple Terminal Manager

Explanation: Initialization was unsuccessful. This message is written to the terminal that loaded the Multiple Terminal Manager. Additional messages are printed on the Multiple Terminal Manager log device.

System Action: Multiple Terminal Manager terminates.

Response: Determine the cause of the error and take corrective action.

Messages

INVALID COMMAND

Issued By: $IAMUT1

Explanation: An invalid command was entered by the user.

System Action: Reprompts for command.

Response: Enter a question mark (?) to obtain a list of valid commands and try again.

INVALID PROGRAM NAME

Issued By: Multiple Terminal Manager

Explanation: The name of the program requested from the primary menu was not found in the Multiple Terminal Manager program table or invalid parameters were supplied on a DISCONNECT command.

System Action: The requested function is not performed.

Response: Correct the program name or parameters and retry the request.

INVALID SIGNON CHARACTER

Issued By: Multiple Terminal Manager

Explanation: The SIGNON specification for the TERMINAL file record listed immediately before this message is not "Y" or "N".

System Action: The terminal is not connected.

Response: Correct the TERMINAL record. Stop and restart the manager.

INVALID TERMINAL

Issued By: Multiple Terminal Manager

Explanation: The terminal name entered with a DISCONNECT command is not a Multiple Terminal Manager terminal.

System Action: The terminal is not disconnected.

Response: Retry specifying a valid terminal name.

KEY OF INPUT REC xxxxxx IS DUPLICATE OR OUT OF SEQUENCE. OMIT THE RECORD AND CONTINUE?

Issued By: $IAMUT1

Explanation: A duplicate key exists in the input sequential data set and could not be written to indexed data set.

System Action: If reply = Y, the next record will be read from input data set and processing continues. If reply = N, command terminates.

Response: Make sure the input data set contains the proper data. Check the KEYSIZE and KEYPOS parameters used to define the indexed data set against the input data set records. Make sure the data in the input data set is in the proper sequence. Redefine and reload data set if necessary.

LOAD ERROR RC=xxx

Issued By: Multiple Terminal Manager

Explanation: A load failure occurred.

System Action: The terminal is not available to the Multiple Terminal Manager.

Response: Check the LOAD return code to find the cause of the problem and take the appropriate action.

Messages

LOAD FOR SERVER xxxxxxxx FAILED, RC=xxx

Issued By: Multiple Terminal Manager

Explanation: A load failure occurred during initialization for the server for terminal xxxxxxxx.

System Action: The terminal is not available to the &m..

Response: Check the LOAD return code to find the cause of the problem and take the appropriate action.

MENUNAME INVALID

Issued By: Multiple Terminal Manager

Explanation: The primary menu name specified for the TERMINAL file record listed immediately before this message is invalid.

System Action: The terminal is not connected.

Response: Correct the TERMINAL record. Stop and restart the manager.

MULTIPLE TERMINAL MANAGER SYSTEM FAILURE

Issued By: Multiple Terminal Manager

Explanation: The Multiple Terminal Manager task error exit routine has been entered due to a machine or program error.

The PSW and LSB at the time of failure has been saved at a displacement of X'172' into the program storage. Register 1 in the LSB contains the address of the failing instruction in the case of a program check.

The following example shows a specification check which occurred at location X'053C'.

```
MULTIPLE TERMINAL MANAGER SYSTEM FAILURE > $A

PROGRAMS  AT  00:06:24  IN  PARTITION #2 $MTM      0000 *
CDMSVR33 6C00 > $D 0 172 30 X  0172: 8002 28E6 0110 10D0
0DDC 053C 0DAC 7361
 0182: 0540 815C 00B8 0DDA 0000 00FA 0004 0028
 0192: 0052 007C 00A6 0017 0E72 A0A2 0E72 FFFF
 01A2: 0102 8026 1616 40C9 D5C9 E3C9
 ANOTHER DISPLAY?
```

The PSW is 8002 at 0172 and R1 is 053C on same line.

System Action: The Multiple Terminal Manager program remains active waiting for an event which will not be posted.

Response: Use Event Driven Executive operator facilities to display storage.

MULTIPLE TERMINAL MANAGER TERMINAL FILE RECORDS

Issued By: Multiple Terminal Manager

Explanation: The TERMINAL file records processed by the Multiple Terminal Manager are listed after this message. Any messages pertaining to a specific TERMINAL file record will be displayed immediately after the file record.

System Action: TERMINAL file records processed are listed.

Response: Review the listing and take action as needed.

Messages

MTMSTORE DATA SET LIMITS EXCEEDED

Issued By: Multiple Terminal Manager

Explanation: The specified MTMSTORE file is too small. This can occur after adding a new program with a storage requirement greater than any previous program's requirement or after adding a new terminal or screen.

System Action: The manager terminates.

Response: Delete the MTMSTORE file and recreate it with more space.

NO BUFFER SUPPLIED. $IAMUT1 TERMINATING

Issued By: $IAMUT1

Explanation: The $STORAGE field in program header was set to 0 and no buffer exists for $IAMUT1.

System Action: The program terminates.

Response: Use the SS command of $DISKUT2 to set the $STORAGE field to the desired buffer size (must be > 0). For LO,UN,and RO commands, this buffer must be large enough to contain the entire input and/or output record (whichever is larger).

NO PROGRAM LOAD FACILITY

Issued By: Load Utility Program

Explanation: The load utility program ($LOADER) cannot be found on the IPL volume.

System Action: The Load utility is terminated.

Response: If $LOADER is not on the IPL volume, you must copy $LOADER from XS-3001 to the IPL volume and restart the Load utility.

NO TERMINALS ARE AVAILABLE

Issued By: Multiple Terminal Manager

Explanation: No valid terminal specification records were found in the TERMINAL file, or, no terminal servers can be loaded, or, all terminals are busy. Other messages generated indicate the problem area.

System Action: The manager terminates.

Response: Determine the cause of the problem and take corrective action.

NO TRAP CONDITIONS SPECIFIED.
$TRAP TERMINATED

Issued By: $TRAP Utility

Explanation: No trap conditions were specified. Some are required.

System Action: The $TRAP utility is terminated.

Response: Specify the necessary trap conditions and restart the $TRAP utility.

NULL INVALID FOR PARAMETER

Issued By: $IAMUT1

Explanation: An attempt was made to specify a null response (&) to a parameter on which this is invalid.

System Action: Reprompts for parameter.

Response: Enter the proper response to the parameter prompt. See determining data set size and format section in the $IAMUT1 chapter of the utilities manual for a description of each parameter. A null response is only valid for RSUBCK, RSUIX, FPOOL AND DELTHR parms.

Messages

OPEN FOR LOAD RETURN CODE = xxx. RETRY ?:

Issued By: $IAMUT1

Explanation: $IAMUT1 attempted to open the specified IAM file in LOAD mode and a bad return code was received from the IAM request.

System Action: If retry = Y, reprompts for DSNAME, VOLUME and retries the IAM open request. If retry = N, command terminates.

Response: Check the Indexed Access Method return code to find the cause of the problem and take the appropriate action.

OPEN FOR PROCESS RETURN CODE = xxx. RETRY?:

Issued By: $IAMUT1

Explanation: $IAMUT1 attempted to open the specified IAM file in process mode and a bad return code was received from the IAM request.

System Action: If retry = Y, reprompts for DSNAME, VOLUME and retries the IAM open request. If retry = N, the command terminates.

Response: Check the Indexed Access Method return code to find the cause of the problem and take the appropriate action.

PARTITION NUMBER IS INVALID

Issued By: $DUMP Utility

Explanation: An invalid partition number was entered during a partial storage dump.

System Action: The $DUMP utility is terminated.

Response: Enter a valid partition number and restart the $DUMP utility.

PRIMARY MENU mmmmmmmm FAILED FOR TERMINAL tttttttt

Issued By: Multiple Terminal Manager

Explanation: A SETPAN function failed for the terminal tttttttt using the primary menu mmmmmmmm.

System Action: The primary menu is not displayed.

Response: Ensure that a valid menu name is specified in the TERMINAL file for the specified terminal.

PROGRAM AREA TOO SMALL TO HOLD PGM pppppppp

Issued By: Multiple Terminal Manager

Explanation: The manager's program area is too small to hold the named program.

System Action: The program is not used.

Response: Increase the program area size by reallocating CDMDUMMY or split the program into smaller link-edited programs.

PROGRAM CAPACITY EXCEEDED

Explanation:

The amount of working storage allocated to $VERIFY is insufficient to process the indexed data set specified.

System Action:

$VERIFY terminates.

Response:

Increase the amount of working storage available to $VERIFY. Refer to Modifying Working Storage Requirements for a description of how to calculate the amount of working storage required and how to modify the amount supplied.

Messages

PROGRAM FILE LARGER THAN PROGRAM MANAGER BUFFER

Issued By: Multiple Terminal Manager

Explanation: The program table built during initialization exceeds the size of the buffer used by the program manager.

System Action: Multiple Terminal Manager terminates.

Response: Increase the program manager buffer size in module CDMCOMMN.

PROGRAM LOAD ERROR

Issued By: Multiple Terminal Manager

Explanation: An Event Driven Executive LOAD error occurred for the requested program.

System Action: The program is not loaded.

Response: Determine the cause of the problem. Rebuild the program if the problem persists.

PROGRAM xxP,00.00.00,LP=zzzz

Issued By: Program Load

Explanation: Any program invoked using $L (Load a Program) results in this message being displayed, indicating that the program you requested has been loaded. Here, xxP indicates that the program is xx pages long (256 bytes equals one page). 00.00.00 is the time in hours, minutes and seconds. LP=xxxx indicates that the load point of the program is at location X'zzzz'. If the timer support is not included in the supervisor, the time is not printed.

System Action: None.

Response: None.

READ INPUT DATASET RETURN CODE = xxx. RECORD NUMBER = xxxxxx

Issued By: $IAMUT1

Explanation: An attempt to read the indexed input data set failed for a sequential data set.

System Action: Command terminates.

Response: LO command - check the READ/WRITE return codes to find the cause of the problem and take the appropriate action. UN, RO commands -check the Indexed Access Method return codes find the cause of the problem and take the appropriate action.

RECONNECT SYNTAX INVALID

Issued By: Multiple Terminal Manager

Explanation: The correct syntax was not used on the RECONNECT operator command.

System Action: The command is ignored.

Response: Retry the RECONNECT command with correct syntax.

RECONNECT TERMINAL DEFINITION ERROR

Issued By: Multiple Terminal Manager

Explanation: The RECONNECT operator interface facility has encountered a failure while attempting to reconnect a terminal to the Multiple Terminal Manager. Since initialization would have already performed all functions necessary to include the terminal in the terminal table, the TERMINAL file, SCRNS volume or source table in RECONNEC has probably been altered since the Multiple Terminal Manager was started.

System Action: Terminal is not connected.

Response: Determine the cause of the error (check TERMINAL file for correct data).

Messages

SCREEN TABLE LARGER THAN INPUT BUFFER

Issued By: Multiple Terminal Manager

Explanation: The screen table built during initialization exceeds the Input Buffer size.

System Action: Initialization is aborted.

Response: Increase the Input Buffer size in module CDMCOMMN.

SENSOR I/O DEVICE AT ADDRESS xxxx IS OFFLINE
BSCA NOT THE DEVICE AT ADDR: zzzz

Issued By: Sensor I/O Status Check

Explanation: The system checks the status of any defined sensor I/O or Binary Synchronous Communications Adapter devices and prints appropriate status messages.

System Action: None.

Response: None.

SET DATE AND TIME USING COMMAND $T

Issued By: System

Explanation: If timer support was included during system generation, the system prints a message indicating that the date and time can be optionally entered (or reset) using the $T supervisor utility.

System Action: None.

Response: None.

SIGNON PROGRAM NOT AVAILABLE FOR TERMINAL ttttttt

Issued By: Multiple Terminal Manager

Explanation: The specified terminal is required to sign on and off but no program named SIGNON was found in the PRGRMS volume.

System Action: The terminal is not signed on.

Response: Place a program named SIGNON in the PRGRMS file or designate that no signon is needed for the specified terminal. Reconnect terminal.

TAPE xxxx IS NOT A TAPE * * *
TAPE xxxx MARKED UNUSABLE

Issued By: Tape Initialization

Explanation: If an address is incorrectly defined (for example, the device is not a tape), if the tape drive is not turned on, or if the tape drive has a hardware failure, messages describing the problem are issued.

System Action: The initialization is terminated.

Response: Correct the problem and restart the initialization.

TAPE xxxx OFFLINE FOR BLP yyyy BPI

Issued By: Tape Initialization

Explanation: If the address (yyyy) is valid and the tape is not mounted this message is printed.

System Action: The initialization is terminated.

Response: Mount the tape and restart the initialization.

Messages

TAPE xxxx TAPE01 ONLINE FOR BLP yyyy BPI

Issued By: Tape Initialization

Explanation: If the address (yyyy) is valid and the tape is mounted this message is printed.

System Action: None.

Response: None.

TERMINAL ttttttt BUSY

Issued By: Multiple Terminal Manager

Explanation: Terminal ttttttt specified in the TERMINAL file is connected to another program.

System Action: The terminal is not used.

Response: Try to RECONNECT at a later time.

TERMINAL ttttttt NOT DEFINED IN EVENT DRIVEN EXECUTIVE SYSTEM

Issued By: Multiple Terminal Manager

Explanation: The specified terminal was not included in the definition of terminals when the Event Driven Executive system was generated.

System Action: The terminal is not connected.

Response: Include a terminal definition for the specified terminal when the Event Driven Executive system is generated.

TERMINAL tttttttt RECONNECTED

Issued By: Multiple Terminal Manager

Explanation: The named terminal has been reconnected to the Multiple Terminal Manager.

System Action: The terminal is reconnected to the Multiple Terminal Manager.

Response: Use the terminal as needed.

TERMINAL NAME INVALID

Issued By: Multiple Terminal Manager

Explanation: The terminal name specified for the TERMINAL file record listed immediately before this message is invalid.

System Action: The terminal is not connected.

Response: Correct the TERMINAL record. Stop and restart the manager.

TERMINAL TABLE OR STORAGE SIZE EXCEEDED

Issued By: Multiple Terminal Manager

Explanation: While building the terminal table and loading servers, the storage size or the the maximum number of terminals (10) allowed has been exceeded. The work space, defined in CDMINIT, is defined to allow a maximum of 50 terminals.

System Action: The extra terminals are not connected.

Response: Increase the terminal table size by changing module CDMCOMMN. If there is not enough room, make the partition larger, decrease the number of terminals, or make CDMDUMMY smaller.

Messages

VALUE OUT OF RANGE

Issued By: $IAMUT1

Explanation: An invalid value was entered for the parameter prompt.

System Action: Reprompt for parameter.

Response: Enter the proper response to the parameter prompt. See Determining Data Set Size and Format section under $IAMUT1 chapter of Utilities manual for a description of each parameter.

VERIFICATION COMPLETE, ___ ERROR(S) ENCOUNTERED

Explanation:

$VERIFY has completed normally.

System Action:

$VERIFY terminates.

Response:

Examine any reports printed as needed.

WRITE OUTPUT DATASET RETURN CODE = xxx. RECORD NUMBER = xxxxxx.

Issued By: $IAMUT1

Explanation: An attempt to write to a sequential (output) data set failed.

System Action: Command terminates.

Response: LO, RO commands - Check the Indexed Access Method return code to find the cause of the problem and take the appropriate action (NOTE1). UN command - Check the READ/WRITE return code to find the cause of the problem and take the appropriate action (NOTE2).

NOTE1: It may be necessary to redefine data set (SE,DF) and retry LO command.

NOTE2: It may be necessary to reallocate the data set and retry the UN command.

xxxxxxxx DISCONNECT

Issued By: Multiple Terminal Manager

Explanation: Terminal xxxxxxxx has been issued a successful DISCONNECT command.

System Action: The terminal is disconnected.

Response: Reconnect terminal as needed.

xxxxxxxx PROGRAM TYPE INVALID

Issued By: Multiple Terminal Manager

Explanation: Program xxxxxxxx in the PRGRMS volume is not a program type data set.

System Action: The program named is not used.

Response: Specify program type members only for use as programs.

Messages

xxxxxxxx SCREEN SIZE TOO LARGE

Issued By: Multiple Terminal Manager

Explanation: Screen xxxxxxxx in the SCRNS volume will not fit in the screen manager buffer.

System Action: The screen is not available during this Multiple Terminal Manager session.

Response: Increase the screen manager buffer size in CDMCOMMN.

xxxxxxxx SETPAN FAILED, RC=xxx

Issued By: Multiple Terminal Manager

Explanation: A SETPAN failed for the screen named xxxxxxxx.

System Action: Processing continues.

Response: Check the Multiple Terminal Manager return code to find the cause of the problem and take the appropriate action.

Program Check Error Message

If a program check occurs during execution of a program, a message with the following format is printed on the loading terminal:

```
PGM CHK:  PLP    TCB     PSW    LSB
          6B00   0138    8002   1E6A   0000   88D0   ...
```

where:

PLP The program load point of the failing program.

TCB The location of the task control block for the failing program (the address appearing on the assembly listing).

PSW The processor status word when the check occurred (described under Processor Status Word)

LSB Level status block, consisting of the following:

```
WORD 1       - instruction address register (IAR)
WORD 2       - address key register (AKR)
WORD 3       - level status register (LSR)
WORD 4 - 11  - general registers (R0-R7)
```

If the program is written in assembler language, COBOL, FORTRAN, or PL/I, the contents of the registers depend upon the conventions unique to that language. If the program is written in Event Driven Language, registers 0 through 7 (words 4-11) contain:

```
WORD  4   -  (R0) work register
WORD  5   -  (R1) address of Event Driven Executive
                  instruction
WORD  6   -  (R2) address of EDL TCB
WORD  7   -  (R3) address of EDL operand 1
WORD  8   -  (R4) address of EDL operand 2
WORD  9   -  (R5) EDL command
WORD 10   -  (R6) work register
WORD 11   -  (R7) work register
```

The program in which the error occurred is either aborted or, if it has a task error exit, the exit is entered. In either case, normal system execution is resumed after the program check message has been printed.

System Program Check Error Message

If a program check occurs in the supervisor, the following message prints on the $SYSLOG terminal:

```
SYSTEM PGM CHK:  PSW AND LSB
8000 0000 1014 80DP 6F00 6F22 1015 54F5 6F26 805C ...
```

where:

```
WORD  1        - processor status word (PSW)
WORD  2        - instruction address register (IAR)
WORD  3        - address key register (AKR)
WORD  4 - 11   - level status register (LSR)
```

Processor Status Word (PSW)

The processor status word (PSW) is used to record error or exception conditions in the system that may prevent further processing. It also contains certain status flags related to error recovery. Error or exception conditions recorded in the PSW cause four of the possible seven class interrupts to occur. These are machine check, program check, soft exception trap, and power/thermal warning.

The Copy Processor Status and Reset (CPPSR) instruction can be used to examine the PSW. This instruction stores the contents of the PSW into a specified location in main storage.

The PSW is contained in a 16-bit register with the following bit representation:

Bit	Processor Type 495x 2	3	5	Condition	Class Interrupt	Note
00	X	X	X	Specification Check	Program Check	
01	X	X	X	Invalid Storage Addr	Program Check	
02	X	X	X	Privilege Violate	Program Check	
03	X		X	Protect Check	Program Check	
		X		Not Used		1
04	X	X	X	Invalid Function	Soft Exception Trap	
05			X	Floating Point Exception	Soft Exception Trap	
	X	X		Not Used		1
06	X	X	X	Stack Exception	Soft Exception Trap	
07	-	-	-	Not Used		1
08	X	X	X	Storage Parity Check	Machine Check	
09	-	-	-	Not Used		1
10	X	X	X	CPU Control Check	Machine Check	
11	X	X	X	I/O Check	Machine Check	
12	X	X	X	Sequence Indicator	None	2
13	X	X	X	Auto IPL	None	2
14	X		X	Translator Enabled	None	
		X		Not Used	-	1
15	X	X	X	Power/Thermal Warning	Power/Thermal	3

Notes:

1. Always Zero

2. Status Flag

3. Controlled by summary mask

Messages

Following is an explanation of the bit representations:

Bit 00 Specification Check: Set to one if (1) the storage address violates the boundary requirements of the specified data type, or (2) the effective address is odd when attempting to execute a floating-point instruction and the floating-point feature is not installed.

Bit 01 Invalid Storage Address: Set to one when an attempt is made to access a storage address outside the storage size of the system. This can occur on an instruction fetch, an operand fetch, or an operand store.

Bit 02 Privilege Violate: Set to one when a privileged instruction is attempted in the problem state (supervisor state bit in the level status register is not on).

Bit 03 Protect Check: In the problem state, this bit is set to one when (1) an instruction is fetched from a storage area not assigned to the current operation, (2) the instruction attempts to access a main storage operand in a storage area not assigned to the current operation, or (3) the instruction attempts to change a main storage operand in violation of the read-only control.

Bit 04 Invalid Function: Set to one by the following conditions:

1. Attempted execution of an illegal operation code or function combination. These are:

Op code	Function
00101	All (when register 7 is specified in the R1 or R2 field of the instruction)
00111	All
01000	0001, 0010, 0011, 0101, 0110, 0111
01011	0001, 1001 (when in supervisor state and the relocation translator feature is not installed)
01011	0101, 0111
01100	111
01110	11000, 11010, 11011, 11100, 11110, 11111
11011	All
10110	All
11101	1100, 1101, 1110, 1111

Note: The preceding illegal conditions cause a program check class interrupt to occur.

2. The processor attempts to execute an instruction associated with a feature that is not installed. These are:

Op code	Function
00100	All (floating-point feature not installed)
01011	0011, 1011 (if the floating-point feature is not installed and the processor is in supervisor state)

Note: The preceding condition causes a soft-exception-trap class interrupt to occur.

Bit 05 Floating-Point Exception: Set to one when an exception condition is detected by the option floating-point processor. The arithmetic indicators (carry, even, and overflow) define the specific condition.

Bit 06 Stack Exception: Set to one when an attempt has been made to pop an operand from an empty main storage stack or push an operand into a full main storage stack. A stack exception also occurs when the stack cannot contain the number of words to be stored by a Store Multiple (STM) instruction.

Bit 08 Storage Parity: Set to one when a parity error has been detected on data being read out of storage by the processor. This error may occur when accessing a storage location that has not been validated since power on.

Bit 10 CPU Control Check: A control check will occur if no levels are active but execution is continuing. This is a machine-wide error. (See I/O check note.)

Bit 11 I/O Check: Set to one when a hardware error has occurred on the I/O interface that may prevent further communication with any I/O device. PSW bit 12 (sequence indicator) is a zero if the error occurred during an Operate I/O instruction and is set to one if the error occurred during a non-DPC transfer. The sequence indicator bit is not an error in itself but reflects the last interface sequence at any time. An I/O check cannot be caused by a software error. (See note.)

Note: The machine check class interrupt initiated by a CPU control check or I/O check causes a reset. The I/O channel and all devices in the system are reset as if a Halt I/O (channel directed command) had been executed. The processor, sensor-based output points, and timer values are not reset.

Bit 12 Sequence Indicator: This bit reflects the last I/O interface sequence to occur. See I/O check described above.

Messages

Bit 13 Auto IPL: Set to one by hardware when an automatic IPL occurs.

Set to zero by:

- A power on reset when Auto IPL mode is not selected
- Pressing the Load key
- An IPL initiated by a host system

Refer to the appropriate hardware manual for a description of initial program load.

Bit 14 Translator Enabled: When the Storage Address Relocation Translator Feature is installed, this bit is set to one or zero as follows:

1. Set to one (enabled)
   - An Enable (EN) instruction is executed with bit 12 of the instruction word set to zero and bit 14 set to one
2. Set to zero (disabled)
   - A Disable (DS) instruction is executed with bit 14 of the instruction word set to one
   - An Enable (EN) instruction is executed with bit 12 of the instruction word set to one
   - A processor reset (power-on reset, check restart, IPL, or system reset key)

Bit 15 Power Warning and Thermal Warning: Set to one when these condition occur (refer to the appropriate hardware manual for a description of a Power/Thermal Warning class interrupt). The power/thermal class interrupt is controlled by the summary mask.

For a description of class interrupts, I/O interrupts and the basic instruction set (including indicator settings and possible exceptions conditions) for your specific processor, refer to the appropriate hardware manual.

Completion Codes

CODES

This section presents three types of codes issued by the Event Driven Executive:

Completion Issued by utility programs upon completion to indicate if execution was successful or not

Return Issued as the result of executing an Event Driven language instruction or subroutine to indicate success or failure of the operation

Post Issued by the system to signal the occurrence of an event

The codes and their meanings are presented by type and alphabetically by functional grouping.

Utility Completion Codes

The completion codes and their meanings are presented in alphabetic order according to function as follows:

- $EDXASM
- $IAMUT1
- $JOBUTIL
- $LINK
- $UPDATE

The utility completion codes are printed on the specified list device by the utility programs upon their completion unless otherwise noted.

$EDXASM Completion Codes

$EDXASM completion codes are accompanied by an appropriate error message and appear at the end of the $EDXASM listing. The completion codes can be tested by the job stream processor, allowing steps subsequent to the assembly to be skipped, if appropriate. The completion codes are:

Completion Code	Condition
-1	Successful completion - no errors in assembly
8	Successful completion - one or more statements had assembly errors
12	Out of space in work or object data set
12	I/O error in source, work, or object data set
12	Overlay-instruction table full
12	Unable to locate overlay program or copy code module
100	Operator cancelled assembly with ATTN CA command

Completion Codes

$IAMUT1 Completion Codes (Part 1 of 2)

Completion Code	Condition
-1	Successful completion
01	Data set not found (OPEN failed)
02	Invalid IODA exit (OPEN failed)
03	Volume not mounted (OPEN failed)
04	Library not found (OPEN failed)
05	Disk I/O error (OPEN failed)
06	No VTOC exit address (OPEN failed)
07	Link module in use
08	Load error for $IAM
12	Data set shut down
13	Module not included in load module
23	Get storage error - IACB
31	FCB WRITE error during IDEF processing
32	Blocksize not multiple of 256
34	Data set is too small
36	Invalid block size during file definition processing
37	Invalid record size
38	Invalid index size
39	Record size greater than block size
40	Invalid number of free records
41	Invalid number of clusters
42	Invalid key size
43	Invalid reserve index value
44	Invalid reserve block value
45	Invalid free pool value
46	Invalid delete threshold value
47	Invalid free block value
48	Invalid number of base records
49	Invalid key position
50	Data set is opened for exclusive use
51	Data set opened in load mode
52	Data set is opened, cannot be opened exclusively
54	Invalid block size during PROCESS or LOAD
55	Get storage for FCB error

$IAMUT1 Completion Codes (Part 2 of 2)

Completion Code	Condition
56	FCB READ error
60	LOAD mode key is equal to or less than previous high key in data set
61	End of file
62	Duplicate key found
100	READ error
101	WRITE error
110	WRITE error - data set closed

Completion Codes

$JOBUTIL Completion Codes

The $JOBUTIL completion codes are displayed on the terminal used to access $JOBUTIL. The codes are as follows.

Completion Code	Condition
-1	Successful completion
61	The transient loader ($LOADER) is not included in the system
64	No space available for the transient loader
67	A disk or diskette I/O error occurred during the load process
70	Not enough main storage available for the program
71	Program not found on the specified volume
72	Disk or diskette I/O error while reading directory
73	Disk or diskette I/O error while reading program header
74	Referenced module is not a program
75	Referenced module is not a data set
76	Data set not found on referenced volume
77	Invalid data set name
78	LOAD instruction did not specify required data set(s)
79	LOAD instruction did not specify required parameter(s)
80	Invalid volume label specified; for example, greater than eight characters

$LINK Completion Codes (Part 1 of 3)

Comp. Code	Condition	Cause code	Action code	Retur code
-	Successful completion	-	-	-1
01	DS2 less than 265 records	1	2	12
02	Disk error reading DS1	2	2	12
03	End of file reached on DS1	1	3	4
04	Disk error reading object module	2	1	8
05	Invalid 'OUTPUT' record	1	2	12
06	Invalid 'INCLUDE' record	1	6	8
07	Error opening object output module: - misspelled name or volume - data set not allocated	1	5	12
08	Error opening input object module (see Error 07)	1	6	8
09	Error opening output module (hardware error)	2	5	12
10	Error opening an input module (hardware error)	2	6	8
11	Error opening autocall list (DS9). See Error 07 for causes	1	5	12
12	Error opening autocall list (DS9) (hardware error)	2	5	12
13	Invalid input object module record type	4	4	8
14	Entry point label not found	1	3	4
15	No valid ESDID for TXT or RLD	4	4	8
16	Invalid ESD item type	4	4	8
17	Duplicate ESDID number	4	4	8
18	Invalid Symbol	4	4	8
19	Duplicate Entry point symbol	3	4	8
20	Invalid ESDID number	4	4	8
22	Invalid ESD symbol	4	1	8

Completion Codes

$LINK Completion Codes (Part 2 of 3)

Comp. Code	Condition	Cause code	Action code	Return code
23	End of file reached on DS9	1	2	12
24	Disk error reading DS9	2	2	12
25	Disk error reading DS4	2	2	12
26	End of File reached on DS3	6	2	12
27	Disk error Read/Write on DS8	2	2	12
28	End of file reached on DS8	5	2	12
29	End of file reached on DS7	6	2	12
30	End of file reached on DS4	6	2	12
31	Disk error writing on DS5	2	2	12
32	End of file reached on DS5	5	2	12
33	End of file reached on DS2	6	2	12
34	Duplicate section definition (CSECT)	3	1	4
36	End of file reached on DS6	4	1	8
37	Disk error, read/write on DS7	2	2	12
38	Disk error, read/write on DS3	2	2	12
39	Invalid RLD record data length	4	4	8
40	Disk error, read/write on DS2	2	2	12
42	DS2 not large enough (program size over 64K)	5	2	12
45	No 'INCLUDE' records	1	2	12
46	No CSECT length field	4	3	4
None	Unresolved EXTRN			4

$LINK Completion Codes (Part 3 of 3)

Cause Codes

1 - Your error
2 - System error
3 - Possible duplicate 'name,volume' or duplicate CSECT or ENTRY names
4 - Input object record(s) in error. Probable cause is that 'name,volume' is not a valid object module
5 - Data set is of insufficient size
6 - Probable $LINK error, this condition should not occur

Action Codes

1 - Log warning message and continue at next 'INCLUDE'
2 - Terminate $LINK with error message
3 - Continue as if expected occurance had happened
4 - Log error message plus invalid object module record and continue at next 'INCLUDE'
5 - Log error message plus OUTPUT record and terminate $LINK
6 - Log error message plus INCLUDE record, continue at next 'INCLUDE'

Return Code Definitions

-1 Successful completion
4 Warning: A module has been written - execution will probably work
8 Warning: A module has been written - execution will probably fail
12 Severe error: Module is not written

Completion Codes

$UPDATE Completion Codes

The $UPDATE completion codes are displayed on the terminal used to access $UPDATE. The codes are as follows:

Completion Code	Condition
-1	Successful completion
8	No supervisor space in this library
8	Output name specified is not a program
8	Disk volume already in use by another program
8	No space in directory
8	No space in data set (output library)
8	Invalid header format
8	Invalid program name
8	Disk volume not mounted
8	Disk volume off line
8	Library not found
8	Input data set not found
8	No parameter supplied via $JOBUTIL
8	No data set names provided via $JOBUTIL
8	Replacement of output data set not allowed
12	Any disk or diskette I/O errors

EVENT DRIVEN LANGUAGE AND FUNCTION RETURN CODES

The return codes and their meanings are presented in alphabetic order according to function as follows:

- $DISKUT3
- $PDS
- BSC
- Data Formatting
- Disk and Tape (READ/WRITE)
- EXIO
- Floating-point
- Formatted Screen Image as follows:
  - $IMDATA subroutine
  - $IMOPEN subroutine
  - $IMPROT subroutine
- Indexed Access Method
- Multiple Terminal Manager
- SBIO (Sensor Based I/O)
- Terminal I/O as follows:
  - General
  - ACCA
  - Interprocessor Communications
  - Virtual Terminal
- TP (Host Communication Facility)

The return codes are issued by EDL instructions and EDL-invokable functions. They are returned in the first word of the task control block of the calling program unless otherwise noted.

Return Codes

$DISKUT3 Return Codes

The $DISKUT3 utility places a return code in the first word of the DSCB specified. The return codes for $DISKUT3 are listed below.

Return Code	Condition
1	Invalid request code parameter (not 1-6)
2	Volume does not exist (All functions)
4	Insufficient space in library (ALLOCATE)
5	Insufficient space in directory (ALLOCATE)
6	Data set already exists - smaller than the requested allocation
7	Insufficient contiguous space (ALLOCATE)
8	Disallowed data set name, eg. $EDXVOL or $EDXLIB (All functions)
9	Data set not found (OPEN, RELEASE, RENAME)
10	New name pointer is zero (RENAME)
11	Disk is busy (ALLOCATE, DELETE, RELEASE, RENAME)
12	I/O error writing to disk (ALLOCATE, DELETE, RELEASE, RENAME)
13	I/O error reading from disk (All functions)
14	Data set name is all blanks (ALLOCATE, RENAME)
15	Invalid size specification (ALLOCATE)
16	Invalid size specification (RELEASE)
17	Mismatched data set type (DELETE, OPEN, RELEASE, RENAME)
18	Data set already exists - larger than the requested allocation
19	SETEOD only valid for data set of type 'data'
20	Load of $DISKUT3 failed ($RMU only)
21	Tape data sets are not supported

$PDS Return Codes

The $PDS utility returns the status of an event in the event control block (ECB) specified by the EVENT= parameter on the LOAD instruction. The return codes for $PDS are listed below.

Return Code	Condition
-1	Successful operation
1	Member not found
2	Member already allocated
3	No space
4	Directory is full
5	Member was not used
7	Record not in member
8	Member control block invalid
9	Space not released
10	Not a data member

Return Codes

BSC Return Codes

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

Notes:

1. Retried up to the limit specified in the RETRIES= operand of the BSCLINE definition.

2. Not retried.

3. Retried during write operation only when a wrong ACK is received following an ENQ request after timeout (indicating that no text had been received at the remote station).

4. Returned only during an initial sequence with no retry attempted.

5. Retried only after an unsuccessful start I/O attempt.

6. Retried only during read operations.

Return Codes

Data Formatting Return Codes

Return Code	Description
-1	Successful completion
1	No data in field
2	Field omitted
3	Conversion error

These return codes are issued by the CONVTB, CONVTD, GETEDIT, and PUTEDIT instructions.

Disk and Tape (READ/WRITE) Return Codes

Disk and tape return codes resulting from READ/WRITE instructions are returned in two places:

1. the Event Control Block (ECB) named DSn, where n is the number of the data set being referenced.

2. the task code word referred to by taskname.

The disk and tape return codes and their meanings are shown below.

If further information concerning an error is required, it may be obtained by printing all or part of the contents of the Disk Data Blocks (DDBs) located in the Supervisor. The starting address of the DDBs can be obtained from the linkage editor map of the supervisor. The contents of the DDBs are described in the Internal Design. Of particular value are the Cycle Steal Status Words and the Interrupt Status Word save areas, along with the contents of the word that contains the address of the next DDB in storage.

Return Codes

Disk Return Codes

Return Code	Condition
-1	Successful completion
1	I/O error and no device status present (this code may be caused by the I/O area starting at an odd byte address)
2	I/O error trying to read device status
3	I/O error retry count exhausted
4	Read device status I/O instruction error
5	Unrecoverable I/O error
6	Error on issuing I/O instruction for normal I/O
7	A 'no record found' condition occurred, a seek for an alternate sector was performed, and another 'no record found' occurred, for example, no alternate is assigned
9	Device was 'offline' when I/O was requested
10	Record number out of range of data set--may be an end-of-file (data set) condition
11	Data set not open or device marked unusable when I/O was requested
12	DSCB was not OPEN; DDB address = 0

Note: The actual number of records transferred is in the second word of the TCB.

Tape Return Codes

Return Code	Condition
-1	Successful completion
1	Exception but no status
2	Error reading STATUS
4	Error issuing STATUS READ
5	Unrecoverable I/O error
6	Error issuing I/O command
10	Tape mark (EOD)
20	Device in use or offline
21	Wrong length record
22	Not ready
23	File protect
24	EOT
25	Load point
26	Uncorrected I/O error
27	Attempt WRITE to unexpired data set
28	Invalid blksize
29	Data set not open
30	Incorrect device type
31	Incorrect request type on close request
32	Block count error during close
33	EOV1 label encountered during close
76	DSN not found

Note: The actual number of records transferred is in the second word of the TCB.

Return Codes

EXIO Return Codes (Part 1 of 2)

I/O Instruction Return Codes (word 0 of TCB; word 1 of TCB contains supervisor instruction address)

Return Code	Condition
-1	Command accepted
1	Device not attached
2	Busy
3	Busy after reset
4	Command reject
5	Intervention required
6	Interface data check
7	Controller busy
8	Channel command not allowed
9	No DDB found
10	Too many DCBs chained
11	No address specified for residual status
12	EXIODEV specified zero bytes for residual status
13	Broken DCB chain (program error)
16	Device already opened

EXIO Return Codes (Part 2 of 2)

Interrupt Condition Codes (bits 4-7 of word 0 of ECB)
(If bit 0 is on, bits 8-15=device ID)

Return Code	Condition
0	Controller end
1	Program Controlled Interrupt (PCI)
2	Exception
3	Device end
4	Attention
5	Attention and PCI
6	Attention and exception
7	Attention and device end
8	Not used
9	Not used
10	SE on and too many DCBs chained
11	SE on and no address specified for residual status
12	SE on and EXIODEV specified no bytes for residual status
13	Broken DCB chain
14	ECB to be posted not reset
15	Error in Start Cycle Steal Status (after exception)

Return Codes

Floating-Point Return Codes

Return Code	Description
-1	Successful completion
1	Floating point overflow
3	Floating point divide check (divide by '0')
5	Floating point underflow

Formatted Screen Image Return Codes

These return codes are issued by the $IMDATA, $IMOPEN, and $IMPROT subroutines. They are returned in the second word of the task control block (TCB) of the calling program.

$IMDATA—Screen Image Unprotected Fields

Return Code	Condition
-1	Successful completion
9	Invalid format in buffer

$IMOPEN - Formatted Screen Image

Return Code	Condition
-1	Successful completion
1	Disk I/O error
2	Invalid data set name
3	Data set not found
4	Incorrect header or data set length
5	Input buffer too small
6	Invalid volume name
7	No 3101 image available
8	Data set name longer than eight-bytes

Return Codes

$IMPROT - Screen Image Protected Fields

Return Code	Condition
-1	Successful completion
9	Invalid format in buffer
10	FTAB truncated due to insufficient buffer size
11	Error in building FTAB from 3101 format; partial FTAB created

Indexed Access Method Return Codes

The following identify messages which indicate that the indexed data set contains errors:

Return Code	Condition
-1	Successful completion
-57	Data set has been loaded
-58	Record not found
-80	End of data
-85	Record to be deleted not found
01	Function code not recognized
07	Link module in use
08	Load error for $IAM
10	Invalid request
12	Data set shut down due to error
13	Module not included in load module
22	Invalid IACB address
23	Get storage error - IACB
50	Data set is opened for exclusive use, cannot be opened exclusively
51	Data set opened in load mode
52	Data set is opened, cannot be opened exclusively
54	Invalid block size during PROCESS or LOAD processing
55	Get storage error - FCB
56	READ error - FCB
60	Out of sequence or duplicate key
61	End of file
62	Duplicate key found in PROCESS mode
70	No space for insert
80	FCB WRITE error during DELETE processing
85	Key field modified by user
90	Key save area in use
100	READ error
101	WRITE error
110	WRITE error - data set closed

Return Codes

LOAD Return Codes

Return Code	Condition
-1	Successful completion
61	The transient loader ($LOADER) is not included in the system
62	In an overlay request, no overlay area exists
63	In an overlay request, overlay area is in use
64	No space available for the transient loader
65	In an overlay load operation, number of data sets passed by the LOAD instruction does not equal number required by the overlay program
66	In an overlay load operation, no parameters were passed to the loaded program
67	A disk or diskette I/O error occurred during the load process
68	Reserved
69	Reserved
70	Not enough main storage available for program
71	Program not found on the specified volume
72	Disk or diskette I/O error while reading directory
73	Disk or diskette I/O error while reading program header
74	Referenced module is not a program
75	Referenced module is not a data set
76	One of the data sets not found on referenced volume
77	Invalid data set name
78	LOAD instruction did not specify required data set(s)
79	LOAD instruction did not specify required parameters(s)
80	Invalid volume label specified
81	Cross partition LOAD requested, support not included at system generation
82	Requested partition number greater than number of partitions in the system

Note: If the program being loaded is a sensor I/O program and a sensor I/O error is detected, the return code will be a sensor I/O return code, not a load return code.

Multiple Terminal Manager Return Codes

These return codes are returned in a caller-specified variable on the SETPAN, FILEIO, FTAB, or SETFMT function.

CODE	DESCRIPTION
-501	Screen data set not found
-500	Terminal is not an IBM 4978/4979 or 3101; no action has been taken
-2	FTAB code not link edited with application
-1	Successful completion
1	Warning: For SETPAN, this is an uninitilized panel. Input buffer has been set to unprotected blanks (x'00') and cursor position set to zero. For FTAB, no fields were found.
2	For SETPAN, unprotected data is truncated. For FTAB, the FTAB table is truncated. For SETFMT, data stream is truncated.
3	No data stream found
201	Data set not found
202	Volume not found
203	No file table entries are available; all have updates outstanding
204	I/O error reading volume directory
205	I/O error writing volume directory
206	Invalid function request
207	Invalid key operator
208	SEOD record number greater than data set length
Other	Return code from READ/WRITE or the Indexed Access Method

Return Codes

SBIO (Sensor-based I/O) Return Codes

Return Code	Condition
-1	Successful completion
90	Device not attached
91	Device busy or in exclusive use
92	Busy after reset
93	Command reject
94	Invalid request
95	Interface data check
96	Controller busy
97	Analog Input over voltage
98	Analog Input invalid range
100	Analog Input invalid channel
101	Invalid count field
102	Buffer previously full or empty
104	Delayed command reject

Terminal I/O Return Codes

These codes are returned by the PRINTEXT, READTEXT, and TERMCTRL instructions. The codes differ depending on the type of terminal being accessed. Separate tables show general codes, ACCA, General Purpose Interface Bus, Interprocessor Communications, Series/1 to Series/1, and Virtual Terminal return codes.

Terminal I/O - General

Return Code	Condition
-1	Successful completion
1	Device not attached
2	System error (busy condition)
3	System error (busy after reset)
4	System error (command reject)
5	Device not ready
6	Interface data check
7	Overrun received
>10	Codes greater than 10 represent possible multiple errors. To determine the errors, subtract 10 from the code and express the result as an 8-bit binary value. Each bit (numbering from the left) represents an error as follows:
-	Bit 0 - Unused
-	Bit 1 - System error (command reject)
-	Bit 2 - Not used
-	Bit 3 - System error (DCB specification check)
-	Bit 4 - Storage data check
-	Bit 5 - Invalid storage address
-	Bit 6 - Storage protection check
-	Bit 7 - Interface data check

Note: For 2741 or PROC devices, subtract 128, not 10; the result then contains status word 1 of the ACCA. (Refer to Communication Features Description to determine the special error condition.)

Return Codes

Terminal I/O - ACCA Return Codes

Return Code	Condition
-1	Successful completion
Bit	**Condition**
0	Unused
1-08	ISB of last operation (I/O complete)
9-10	Unused
11	1 if a write of control operation (I/O complete)
12	Read operation (I/O complete)
13	Unused
14-15	Condition code +1 after I/O start or condition code after I/O complete

Terminal I/O - Interprocessor Communications Return Codes

	CODTYPE=		
Return Code	EBCD/CRSP	EBCDIC	Condition
-2	1F	FDFF	End of transmission (EOT)
-1	5B	FEFF	End of record (NL)
Handled by device support	Not used	FCFF	End of subrecord (EOSR)

Return Codes

Terminal I/O - Virtual Terminal Communications Return Codes

Value	Transmit	Receive
x'8Fnn'	NA	LINE=nn received
x'8Enn'	NA	SKIP=nn received
-2	NA	Line received (no CR)
-1	Successful completion	New line received
1	Not attached	Not attached
5	Disconnect	Disconnect
8	Break	Break

LINE=nn (x'8Fnn'): This code is posted for READTEXT or GETVALUE instructions if the other side sent the LINE forms control operation; it is transmitted so that the receiving program may reproduce on a real terminal (for printer spooling applications for example) the output format intended by the sending program.

SKIP=nn (x'8Enn'): The sending program transmitted SKIP=nn.

Line Received (-2): This code indicated that the sending program did not send a new line indication, but that the line was transmitted because of execution of a control operation or a transition to the read state. This is how, for example, a prompt message is usually transmitted with READTEXT or GETVALUE.

New Line Received (-1): This code indicates transmission of the carriage return at the end of the data. The distinction between a new line transmission and a simple line transmission is, again, made only to allow preservation of the original output format.

Not attached (1): If the virtual terminal accessed for the operation does not reference another virtual terminal, then this code is returned.

Disconnect (5): This code value corresponds to the 'not ready' indication for real terminals; its specific meaning for virtual terminals is that the program at the other end of the channel terminated either through PROGSTOP or operator intervention.

Break (8): The break code indicates that the other side of the channel is in a state (transmit or receive) which is incompatible with the attempted operation. If only one end of the chan-

nel is defined with SYNC=YES (on the TERMINAL statement), then the task on that end will always receive the break code, whether or not it attempted the operation first. If both ends are defined with SYNC=YES, then the code will be posted to the task which last attempted the operation. The break code may thus be understood as follows: when reading (READTEXT or GETVALUE), the other program has stopped sending and is waiting for input; when writing (PRINTEXT or PRINTNUM), the other program is also attempting to write. Note that current Event Driven Executive programs, or future programs which do not interpret the break code, must always communicate through a virtual terminal which is defined with SYNC=NO (the default).

Return Codes

TP (Host Communication Facility) Return Codes (Part 1 of 3)

Return Code	Condition	Module
-1	Successful completion	Supervisor
1	Illegal command sequence	Supervisor
2	TP I/O error	Supervisor
3	TP I/O error on host	HCFCOMM
4	Looping bidding for the line	Supervisor
5	Host acknowledgement to Supervisor request code was neither ACK0, ACK1, WACK, or a NACK	
6	Retry count exhausted - last error was a timeout: the host must be down	Supervisor
7	Looping while reading data from the host	Supervisor
8	The host responded with other thanan EOT or an ENQ when an EOT was expected	Supervisor
9	Retry count exhausted - last error was a modem interface check	Supervisor
10	Retry count exhausted - last error was not a timeout, modem check, block check, or overrun	Supervisor
11	Retry count exhausted - last error was a transmit overrun	Supervisor
50	I/O error from last I/O in DSWRITE	DSCLOSE
51	I/O error when writing the last buffer	DSCLOSE
100	Length of DSNAME is zero	HCFCOMM
101	Length of DSNAME exceeds 52	HCFCOMM
102	Invalid length specified for I/O	HCFINIT

TP (Host Communication Facility) Return Codes (Part 2 of 3)

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

Return Codes

TP (Host Communication Facility) Return Codes (Part 3 of 3)

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

EVENT DRIVEN LANGUAGE AND FUNCTION POST CODES

The Event Driven language and function post codes are returned to the first word of the event control block (ECB) (unless stated otherwise) to signal the occurrence of an event.

Post Codes

Tape Post Codes

If you initialize a tape by loading $TAPEIT from a program or by invoking $JOBUTIL and passing the above parameters, the following post codes are returned to the event control block (ECB) of the calling program.

Post Code	Condition
-1	Function successful
RC	Any tape I/O return code
101	TAPEID not found
102	Device no offline
103	Unexpired data set on tape
104	Cannot initialize BLP tapes