PERKIN-ELMER

OS/32 MULTI-TERMINAL MONITOR (MTM)

Reference Manual

48-043 F00 R01

The Perkin-Elmer Corporation, Data Systems Group, 2 Crescent Place, Oceanport, New Jersey 07757

TABLE OF CONTENTS

PREFACE		ix
CHAPTERS		
1	GENERAL DESCRIPTION	
1.1	INTRODUCTION	1-1
1.2	MULTI-TERMINAL MONITOR (MTM) OPERATION	1-1
1.3	USER INFORMATION	1-2
1.3.1	Multi-Terminal Monitor (MTM) Devices	1-3
1.3.2	Authorization	1-3
1.3.3	Privileged Users	1-3
1.3.4	Transmitting Messages	1-4
1.3.5	Number of Terminal Users	1-4
1.4	MULTI-TERMINAL MONITOR (MTM) ENVIRONMENTS	1-4
1.4.1	Multi-Terminal Monitor (MTM) Terminal Modes	1-6
1.4.1.1	Interactive Task to Terminal Mode	1-6
1.5	LOADING A TASK	1-7
1.6	MULTI-TERMINAL MONITOR (MTM) SPECIAL FEATURES	1-7
1.6.1	Command Substitution System (CSS)	1-7
1.6.2	The Help Facility	1-7
1.6.3	Program Development Commands	1-8
1.6.4	Spooling	1-8
1.6.5	Security and Access Protection of Disks	1-8
1.6.6	Signon Command Substitution System (CSS)	1-8
1.7	CONVENTIONS	1-9
1.7.1	Prompt Conventions	1-9
1.7.2	Terminal Conventions	1-10
1.7.2.1	Using the Break Key	1-10
1.7.3	Command Conventions	1-11
1.7.4	File Conventions	1-11
1.7.4.1	Private Account Numbers	1-11
1.7.4.2	Group Account Numbers	1-11
1.7.4.3	System Account Numbers	1-12
1.7.4.4	File Descriptors (fds)	1-12

CHAPTERS (Continued)

2 MULTI-TERMINAL MONITOR (MTM) USER COMMANDS

	2.1	INTRODUCTION	2-1
¦	2.2	ALLOCATE COMMAND	2-2
¦	2.3	ASSIGN COMMAND	2-6
	2.4	BFILE COMMAND	2-12
	2.5	BIAS COMMAND	2-13
	2.6	BREAK COMMAND	2-14
	2.7	BRECORD COMMAND	2-15
¦	2.8	BUILD AND ENDB COMMANDS	2-16
	2.9	CANCEL COMMAND	2-18
	2.10	CLOSE COMMAND	2-19
	2.11	CONTINUE COMMAND	2-20
	2.12	DELETE COMMAND	2-21
¦	2.13	DISPLAY COMMAND	2-22
¦	2.14	DISPLAY ACCOUNTING COMMAND	2-24
¦	2.15	DISPLAY DEVICES COMMAND	2-25
	2.16	DISPLAY DFLOAT COMMAND	2-27
¦	2.17	DISPLAY FILES COMMAND	2-28
	2.18	DISPLAY FLOAT COMMAND	2-34
	2.19	DISPLAY LU COMMAND	2-35
¦	2.20	DISPLAY PARAMETERS COMMAND	2-37
	2.21	DISPLAY REGISTERS COMMAND	2-42
	2.22	DISPLAY TIME COMMAND	2-43
¦	2.23	DISPLAY USERS COMMAND	2-44
¦	2.24	ENABLE COMMAND	2-45
¦	2.25	EXAMINE COMMAND	2-46
	2.26	FFILE COMMAND	2-48

CHAPTERS (Continued)

2.27	FRECORD COMMAND	2-49	
2.28	HELP COMMAND	2-50	¦
2.29	INIT COMMAND	2-52	
2.30	LOAD COMMAND	2-53	¦
2.31	LOG COMMAND	2-55	¦
2.32	MESSAGE COMMAND	2-57	
2.33	MODIFY COMMAND	2-58	
2.34	OPTIONS COMMAND	2-60	¦
2.35	PASSWORD COMMAND	2-61	¦
2.36	PAUSE COMMAND	2-62	
2.37	PREVENT COMMAND	2-63	¦
2.38	PRINT COMMAND	2-64	¦
2.39	PUNCH COMMAND	2-66	¦
2.40	$RELEASE COMMAND	2-68	¦
2.41	RENAME COMMAND	2-70	¦
2.42	REPROTECT COMMAND	2-71	¦
2.43	REWIND AND RW COMMANDS	2-72	
2.44	RVOLUME COMMAND	2-73	¦
2.45	SEND COMMAND	2-76	
2.46	SET GROUP COMMAND	2-77	¦
2.47	SET KEYOPERATOR COMMAND	2-79	¦
2.48	SET PRIVATE COMMAND	2-81	¦
2.49	SIGNOFF COMMAND	2-83	
2.50	SIGNON COMMAND	2-84	¦
2.51	SPOOLFILE COMMAND	2-86	¦
2.52	START COMMAND	2-89	
2.53	TASK COMMAND	2-90	

CHAPTERS (Continued)

2.54	TEMPFILE COMMAND	2-91
2.55	VOLUME COMMAND	2-94
2.56	WFILE COMMAND	2-96
2.57	XALLOCATE COMMAND	2-97
2.58	XDELETE COMMAND	2-100
3	MTM/NON-MTM TASK INTERFACES	
3.1	INTRODUCTION	3-1
3.2	INTERFACING WITH A FOREGROUND TASK	3-1
3.2.1	Programming Details	3-2
3.3	HASP INTERFACE	3-4
3.4	ITC/RELIANCE INTERFACE	3-5
4	PROGRAM DEVELOPMENT	
4.1	INTRODUCTION	4-1
4.2	CREATING A SOURCE PROGRAM	4-1
4.2.1	Creating a Data File	4-4
4.3	EXECUTING A PROGRAM	4-4
4.4	MODIFYING A PROGRAM	4-5
4.5	RE-EXECUTING A MODIFIED PROGRAM	4-5
4.6	EXECUTING MULTIPLE PROGRAMS AS A SINGLE PROGRAM	4-6
4.7	HOW TO RECOVER FROM ERRORS	4-10
4.8	ASSIGNING LOGICAL UNITS	4-10
4.9	PROGRAM DEVELOPMENT COMMANDS	4-11
4.9.1	ADD Command	4-12
4.9.2	COMPILE Command	4-14
4.9.3	COMPLINK Command	4-18
4.9.4	EDIT Command	4-21
4.9.5	ENV Command	4-23
4.9.6	EXEC Command	4-25
4.9.7	LINK Command	4-28
4.9.7.1	Link Sequences	4-29
4.9.8	LIST Command	4-32

CHAPTERS (Continued)

4.9.9	REMOVE Command	4-33	
4.9.10	RUN Command	4-34	¦
4.10	SAMPLE PROGRAM DEVELOPMENT SESSIONS	4-36	¦
5	MULTI-TERMINAL MONITOR (MTM) BATCH PROCESSING		
5.1	INTRODUCTION	5-1	
5.2	BATCH COMMANDS	5-1	
5.2.1	INQUIRE Command	5-3	¦
5.2.2	LOG Command	5-5	¦
5.2.3	PURGE Command	5-7	
5.2.4	SIGNOFF Command	5-8	
5.2.5	SIGNON Command	5-9	¦
5.2.6	SUBMIT Command	5-11	¦
5.3	BATCH JOB SUBMISSION USING THE SPOOLER	5-13	
5.4	ERROR HANDLING	5-13	¦
5.5	BATCH TASK PAUSE OPTION	5-13	¦
5.6	EFFECT OF RESTRICTED DISKS ON BATCH JOBS	5-14	
6	COMMAND SUBSTITUTION SYSTEM (CSS)		
6.1	GENERAL DESCRIPTION	6-1	¦
6.2	CALLING A CSS FILE	6-2	
6.3	USE OF PARAMETERS	6-3	¦
6.4	USE OF KEYWORDS	6-5	¦
6.4.1	Referencing Keywords Within the CSS	6-7	¦
6.5	USE OF VARIABLES	6-9	¦
6.5.1	Types of Variables	6-9	¦
6.5.2	Naming Local or Global Variables	6-10	¦
6.5.3	Naming New Global or New Internal Variables	6-10	¦
6.5.4	CSS Line Expansion	6-12	¦
6.5.5	Reserved Variables	6-13	
6.6	COMMANDS EXECUTABLE WITHIN A CSS FILE	6-13	
6.6.1	(%...%) Character Replacement Command	6-14	¦
6.6.2	$BUILD and $ENDB Commands	6-18	¦
6.6.3	$CLEAR Command	6-20	
6.6.4	$CONTINUE Command	6-21	
6.6.5	$COPY and $NOCOPY Commands	6-22	
6.6.6	$DEFINE Command	6-23	¦
6.6.6.1	File Descriptor Operators	6-24	¦

6.6.6.1.1 ACCOUNT Operator 6-24
6.6.6.1.2 EXTENSION Operator 6-25
6.6.6.1.3 FILENAME Operator 6-26
6.6.6.1.4 VOLUMENAME Operator 6-28
6.6.6.2 Logical Operators 6-29
6.6.6.3 Computation and Conversion Operators 6-30
6.6.6.3.1 DCOMPUTE Operator 6-30
6.6.6.3.2 DHCONVERT Operator 6-32
6.6.6.3.3 HCOMPUTE Operator 6-33
6.6.6.3.4 HDCONVERT Operator 6-34
6.6.6.4 Other Operators 6-35
6.6.6.4.1 CLEAR Operator 6-36
6.6.6.4.2 CURRENT Operator 6-37
6.6.6.4.3 DVOLUMENAME Operator 6-38
6.6.6.4.4 REQUIRED Operator 6-39
6.6.6.4.5 SEARCH Operator 6-40
6.6.6.4.6 STRING Operator 6-43
6.6.7 $EXIT Command 6-45
6.6.8 $FREE Command 6-46
6.6.9 $GLOBAL Command 6-47
6.6.10 $JOB and $TERMJOB Commands 6-48
6.6.11 $LOCAL Command 6-50
6.6.12 $PAUSE Command 6-51
6.6.13 PRIOR Command 6-52
6.6.14 $RELEASE Command 6-53
6.6.15 $SET Command 6-55
6.6.16 SET CODE Command 6-56
6.6.17 $SKIP Command 6-57
6.6.18 $WAIT Command 6-58
6.6.19 $WRITE Command 6-59

6.7 LOGICAL IF COMMANDS 6-60
6.7.1 End of Task Code Testing Commands 6-61
6.7.2 File Existence Testing Commands 6-62
6.7.3 Parameter Existence Testing Commands 6-62
6.7.4 $ELSE Command 6-63

6.8 $GOTO AND $LABEL COMMANDS 6-64

6.9 $IFEXTENSION COMMAND 6-66

6.10 $IFVOLUME COMMAND 6-67

6.11 LOGICAL IF COMMANDS COMPARING TWO ARGUMENTS 6-68
6.11.1 $IF...EQUAL, $IF...NEQUAL Commands 6-69
6.11.2 $IF...GREATER, $IF...NGREATER Commands 6-69
6.11.3 $IF...LESS, $IF...NLESS Commands 6-70

CHAPTERS (Continued)

7 SPOOLING

7.1 INTRODUCTION 7-1

7.2 THE OS/32 SPOOLER 7-1
7.2.1 Input Spooling 7-2
7.2.2 Input Spooling Control Card Statements 7-2
7.2.2.1 The /@ INPUT Control Statement 7-2
7.2.2.2 The /@ SUBMIT Control Statement 7-3
7.2.3 Output Spooling 7-5
7.2.4 Spooling Errors 7-6

7.3 THE SPL/32 SPOOLER 7-7
7.3.1 SPL/32 and MTM Interaction 7-8

APPENDIXES

A MTM COMMAND SUMMARY A-1 |

B PROGRAM DEVELOPMENT COMMAND SUMMARY B-1

C CSS COMMAND SUMMARY C-1 |

D MTM MESSAGE SUMMARY D-1 |

E CSS MESSAGE SUMMARY E-1 |

F PROGRAM DEVELOPMENT MESSAGE SUMMARY F-1

G MTM/NON-MTM TASK INTERFACES G-1 |

G.1 $FOREGROUND TASK INTERFACE MESSAGES G-1 |

G.2 HASP INTERFACE MESSAGES G-2 |

H CONTROL SUMMARY FOR BI-DIRECTIONAL INPUT/OUTPUT CONTROL (BIOC) CRT DRIVER H-1 |

FIGURES

4-1 COMPILE Command Functions in the Language Environment 4-16 |
4-2 COMPILE Command Functions in the Multi-Module Environment 4-17
4-3 COMPLINK Command Functions in the Language Environment 4-19

FIGURES (Continued)

4-4	COMPLINK Command Functions in the Multi-Module Environment	4-20
4-5	EXEC Command Functions in the Language Environment	4-26
4-6	EXEC Command Functions in the Multi-Module Environment	4-27
4-7	LINK Command Functions in the Language Environment	4-30
4-8	LINK Command Functions in the Multi-Module Environment	4-31
4-9	RUN Command Function in the Language Environment	4-35
4-10	RUN Command Function in the Multi-Module Environment	4-35
H-1	Perkin-Elmer Model 1200 Mode Selectors	H-2

TABLES

1-1	MTM PROMPT CONVENTIONS	1-9
1-2	TERMINAL CONVENTIONS	1-10
2-1	ACCESS PRIVILEGE COMPATIBILITY	2-8
2-2	DISPLAY PARAMETERS COMMAND FIELDS	2-37
2-3	TASK OPTION BIT DEFINITIONS	2-38
2-4	WAIT STATUS BIT DEFINITIONS	2-40
4-1	PROGRAM DEVELOPMENT LANGUAGE COMMANDS	4-1
4-2	PROGRAM DEVELOPMENT COMMAND AVAILABILITY	4-8
4-3	PROGRAM DEVELOPMENT DEFAULT VARIABLE SETTINGS AND LOGICAL UNIT ASSIGNMENTS	4-10
4-4	PROGRAM DEVELOPMENT COMMANDS THAT COMPILE, LINK, AND EXECUTE	4-36
H-1	LINE DISPLAY COMBINATIONS	H-4

INDEX Ind-1

PREFACE

The information about the Perkin-Elmer Multi-Terminal Monitor (MTM) in this manual is written for the MTM user and can also be helpful to the system operator and system programmer.

Chapter 1 is a general description of the MTM system, containing information on MTM system requirements, MTM features, and various conventions. Chapter 2 describes MTM user commands. Chapter 3 describes MTM to non-MTM task interfaces that allow users to transfer control of their terminal between MTM and other non-MTM tasks (HASP, ITC/Reliance, Foreground) and return to MTM in an orderly fashion. Chapter 4 describes the program development commands. Chapter 5 describes batch processing under MTM. Chapter 6 describes the command substitution system (CSS) and the CSS commands. Chapter 7 describes spooling and briefly elucidates the two spoolers (OS/32 and SPL/32) available to users of OS/32 and MTM.

Appendix A summarizes the MTM user commands. Appendix B is a summary of the program development commands. Appendix C summarizes the CSS commands. Appendix D is an MTM command message summary. Appendix E is a summary of CSS messages. Appendix F is a summary of program development command messages. Appendix G is a summary of MTM to non-MTM task interface messages. Appendix H is a control summary for the Bidirectional Input/Output Control (BIOC) CRT driver.

Revision R01 of this manual adds two new file types to the ALLOCATE, TEMPFILE, and XALLOCATE commands. New software density selection options are added to the ASSIGN command. MTM now supports up to a maximum of 65,535 accounts. A new type of user (a PRIVILEGED user) is introduced along with new commands (SET GROUP, SET PRIVATE, PRIOR) that enable the privileged user to access any account on the system. A new PASSWORD command has been added to allow users to alter their own password to enhance account security. Two new variable types have been added to MTM's CSS processor: new global and new internal variables. A command to define these new variable types, the $DEFINE command, has been added. Also, a new command to release these new variable types, the $RELEASE command, has been added. MTM also has the capability to use keywords and positional parameters in CSS calls and reference them within CSS routines. A powerful character replacement command (%...%) has been added to enable replacement of characters within CSS lines on a call-by-call basis. New interface protocols between MTM and non-MTM tasks are available.

Changes were made to the DISPLAY DEVICES, DISPLAY FILES, DISPLAY LU, PRINT, and PUNCH commands, and control information for the new BIOC CRT driver is presented. A new command (SPOOLFILE command) has been added to enable users of the new spooler, SPL/32, to request spooling functions at the CSS or terminal level.

This manual is intended for use with the OS/32 R06.2 software release and higher.

For information on the contents of all Perkin-Elmer 32-bit manuals, see the 32-Bit Systems User Documentation Summary.

CHAPTER 1
GENERAL DESCRIPTION

1.1 INTRODUCTION

Multi-Terminal Monitor (MTM) permits several terminal users to share system resources. Each user perceives that a computer is at his or her disposal.

Concurrent access from online terminals is useful during application task development because it reduces turnaround time. Other advantages are that concurrent access can be used to extend the type of data processing at an installation. Using the system-supplied interactive software means that editing, task development, and documentation can be done simultaneously. Furthermore, if the system-supplied interactive tasks are supplemented by customer-written tasks, MTM application becomes limitless, supporting a mixture of terminal users such as clerks, software development, and operations personnel.

1.2 MULTI-TERMINAL MONITOR (MTM) OPERATION

Like all general purpose, multi-access, time sharing systems, MTM requires operations involvement from the installation using it. This involvement includes those functions that accompany MTM when it is tailored to a specific installation along with dynamic functions performed when MTM is operating.

Examples of the MTM tailoring functions are:

- Cataloguing authorized users
- System generation (sysgen)
- Establishing an installation's procedures

Examples of dynamic functions are:

- System console control
- Peripheral device supervision
- Spooled output dissemination

Generally, tailoring functions are performed and maintained by the customer's system support group responsible for making computing facilities available to system users. The dynamic functions are performed by a system operator during system operation and are distinct from those functions performed by terminal users.

The system operator can perform all the functions described in the OS/32 Operator Reference Manual, together with operator functions required to administer MTM. At any time the system operator may be initiating and controlling multiple foreground tasks and one background task while operating MTM.

1.3 USER INFORMATION

Under MTM control, a terminal user can:

- load and execute interactive tasks,
- submit multiple batch job requests,
- perform program development,
- perform program debugging,
- create, edit, and manipulate files,
- build, modify, and execute command streams,
- use spooling functions,
- communicate with other terminal users, and
- communicate with the system operator.

A terminal user is either interacting with MTM itself, via commands, or interacting with tasks supplied with the system or developed by the installation. All of the vendor-supplied language translators can be operated as interactive tasks by a terminal user. Additionally, a terminal user can use the vendor-supplied support software programs such as: OS/32 Edit, OS/32 Copy, and OS/32 AIDS. It is the MTM software that performs multiple online accessibility; e.g., time sharing, resource management, batch scheduling, etc.

The terminal user can be local or remote. The interactive terminals for local users are directly connected to the computer and do not require telecommunication devices. Interactive terminals for remote users require connection via telecommunication equipment and data communications software. Basic data communications supports both dedicated and dial-up telecommunication terminals.

1.3.1 Multi-Terminal Monitor (MTM) Devices

These devices can be used at any local or remote installation:

- Video Display Unit (VDU) 550B
- VDU 1100
- VDU 1200
- VDU 1250
- VDU 1251
- Perkin-Elmer SIGMA 10 terminal
- M33 Teletype
- M35 Teletype
- Nonediting VDU
- Carousel
- Carousel 300 and 300 EFC

1.3.2 Authorization

The user must be authorized to use MTM facilities. During the signon procedure, the user must supply an account number and a password that were previously cataloged within an MTM file called the authorized user file (AUF). The AUF is updated and maintained by an MTM-supplied task that can be initiated only by the system operator. The terminal user can then interact with MTM from a terminal.

1.3.3 Privileged Users

A variety of new capabilities, called privileges, are now available to the MTM user. These privileges are associated with an account through the AUF utility and are thereafter available to any user that signs on to that account. For the purpose of delineation throughout the remainder of this manual, any user that is signed on to an account which has any or all of these new capabilities enabled is called a privileged user.

Privileged users may have, in addition to all standard MTM capabilities, extended MTM capabilities such as:

- display all jobs in the batch queue,
- move between private accounts without knowing passwords (SET PRIVATE Command),
- change group account numbers (SET GROUP Command),
- set the priority of a subsequently loaded task via a private CSS (PRIOR Command),
- interface with a HASP protocol and return to MTM control as desired ($HSP), and
- interface with a foreground task from an MTM terminal and return to MTM control as desired ($FRGND).

For information on the specific privileges available through MTM and the procedures for enabling these privileges on an account basis, refer to the OS/32 Multi-Terminal Monitor (MTM) System Planning and Operator Reference Manual.

1.3.4 Transmitting Messages

MTM can transmit messages between terminal users, between a terminal user and the system operator, and from the system operator to all or designated terminal users.

1.3.5 Number of Terminal Users

An installation can have up to 64 terminal users or 64 concurrent batch streams. The sum of terminal users and batch streams cannot exceed 64.

1.4 MULTI-TERMINAL MONITOR (MTM) ENVIRONMENTS

The MTM terminal user controls a single task at the terminal and has the ability to run jobs through batch streams. Using the facilities provided by MTM, the user can load a task, start the task, and then interact with the task during its execution. MTM provides interactive and batch user environments.

In an interactive environment, the user has the ability to interact with a task executing at the terminal. In this environment, a dialogue is carried on between the user and the task. The interactive task receives user commands and processes them.

Only one interactive task at a time can be initiated by each MTM terminal. However, all interactive tasks initiated by MTM terminal users are executed concurrently. During interactive task execution, a terminal user can direct a command to and receive a response from MTM itself.

In a batch environment, a number of jobs are run under a full set of automated procedures. Once a batch job is accepted for execution, no further interaction takes place with the initiating terminal user. Requests for multiple batch jobs can be submitted by a user, and the same terminal can be used to initiate an interactive task.

Unlike interactive tasks, requests for batch jobs will not necessarily be initiated immediately to MTM. Instead, batch jobs are queued by the system, and then the queue of submitted batch jobs awaiting execution is serviced by the system. The number of batch jobs that can be executing concurrently is specified by the system operator.

A terminal user can request one or more batch jobs to be run. MTM maintains a queue of submitted batch jobs and concurrently processes a number of batch jobs specified during MTM system start-up. A terminal user can monitor the progress of a batch job by interrogating the MTM batch queue. The returned status will be either:

```
    AWAITING EXECUTION
```

or

```
    EXECUTING
```

If a job already has completed execution, the returned status will be:

```
    NO JOBS FOUND
```

1.4.1 Multi-Terminal Monitor (MTM) Terminal Modes

An active terminal is defined to be in one of six terminal modes. The current mode of the terminal determines which, if any, MTM terminal commands can be accepted. Thus, it is important for the terminal user to be aware of the current mode of the terminal. The user terminal is defined to be in one of the following six modes:

- Command mode: No task is loaded, CSS procedure is not executing and BUILD is not in effect. All non-task related commands are accepted. An "*" is the default prompt displayed in this mode.
- Task loaded mode: The task was loaded but was not started, or is paused. An "*" is the default prompt displayed in this mode.
- Task executing mode: A task was started and is executing. If started from a CSS, CSS mode is suspended. A "-" is the default prompt displayed in this mode. If an interactive task was started and a data input is requested by the task, then a ">" is the default prompt displayed to the terminal user.
- CSS mode: A CSS procedure is being built or executed. A "-" is the default prompt displayed in this mode. When a CSS terminates, the terminal returns to command mode and a "*" prompt is output. When BUILD is in effect, a "B>" is the default prompt displayed.
- Foreground task mode: the terminal has been transferred to the control of a foreground task. When the foregound task is completed the terminal will return under control of MTM. MTM commands are not recognized when in the foreground task mode.
- Hasp interface mode: the terminal is interfaced with a HASP task. The hasp mode prompt is a " and all commands entered while in this mode are sent to the specified HASP task.

1.4.1.1 Interactive Task to Terminal Mode

When a task issues an SVC 1 I/O operation to an active terminal that is in task executing mode and a previous I/O operation to that terminal is still pending, MTM treats the I/O as a wait operation. This is of no concern for tasks that do SVC 1 wait I/O. However, users with tasks that issue SVC 1 proceed I/O (read or write) should be aware that MTM suspends the task until the I/O is completed. Then MTM posts an SVC 1 proceed I/O completion trap on the task's task queue and allows the task to continue. Completion trap posting occurs only if the appropriate bit is set in the TSW.

1.5 LOADING A TASK

The dynamic nature of OS/32 memory management guarantees loading of a task irrespective of its size unless the task is greater than the available task memory. If not enough memory is free to load a task, then some other task is temporarily rolled out if roll support is included in the operating system at sysgen time. If MTM is sysgened with roll influence enabled, then MTM continually monitors the state of the roll queue to ensure that rolled out tasks are given the opportunity to be rolled back in. MTM ensures equity for all its terminal operators by assigning all the interactive tasks an equal priority. Batch tasks can have user assigned priorities.

1.6 MULTI-TERMINAL MONITOR (MTM) SPECIAL FEATURES

The following features are designed to make MTM easier and more efficient to use:

- Command substitution system (CSS)
- Help facility
- Program development commands
- Spooling
- Security and access protection of disks
- Signon CSS

1.6.1 Command Substitution System (CSS)

A terminal user can build a command file on a disk. Once built, a simple directive to MTM will cause MTM to obtain its directives from the command file. When invoking the command file, the terminal user can supply parameters to the command file that can be used to dynamically modify command execution. Therefore, a single terminal input can easily initiate complex operations.

1.6.2 The Help Facility

The Help facility provides a user online access to documentation for MTM and program development commands. This information is obtained by entering the HELP command.

1.6.3 Program Development Commands

The program development commands are an integrated set of standard CSS procedures that perform two major functions:

- maintain information that remains constant throughout a development effort, and
- keep files current throughout a development effort in terms of checking source, object, and image modules to ensure that their dates are current.

1.6.4 Spooling

Both input and output spooling are provided for terminal users. Tasks never need to be delayed awaiting card readers, card punching, or line printing because a batch job can be submitted via the Spooler. The job runs unattended and output goes to the Spooler.

1.6.5 Security and Access Protection of Disks

Privately owned disks can be marked non-restricted by the system operator to offer an MTM user complete security and access protection of files. The owner of the disk can restrict or enable access of the disk to other MTM users, the system operator, and non-MTM tasks.

1.6.6 Signon Command Subsitution System (CSS)

MTM users can build a special CSS file, USERINIT.CSS, within their private accounts. The CSS file can contain commands to load and start a terminal session, assign logical units, and specify a language environment. At signon time, MTM searches all online disks within the user's private account for the file USERINIT.CSS and automatically executes it.

1.7 CONVENTIONS

These conventions used by MTM are detailed in the following sections:

- Prompt conventions
- Terminal conventions
- Command conventions
- Statement syntax conventions
- File conventions

1.7.1 Prompt Conventions

A prompt is output to a terminal device to indicate that the MTM system is ready to accept input from the user. The default prompts displayed on the terminal devices are shown in Table 1-1.

TABLE 1-1 MTM PROMPT CONVENTIONS

PROMPT	USE
*	Indicates MTM system is ready to accept a command.
>	Indicates a request for input data.
B>	Indicates a request for input data to be copied to a BUILD file.
-	Indicates that the system is ready to accept a command while an interactive task is active or a CSS is running. A new CSS cannot be initiated at this time. A user can instruct MTM to suppress or enable the appearance of this prompt while an interactive task is running, but not while CSS is running.
"	Indicates that the terminal is in HASP mode.

1.7.2 Terminal Conventions

The conventions in effect for various terminal devices are shown in Table 1-2.

TABLE 1-2 TERMINAL CONVENTIONS

OPERATION	CONVENTION
Delete a line	To delete a line, simultaneously depress the CTRL and character x keys for all terminals except TEC 455 VDU, which uses the number sign (#). Basic communications support both # and CTRL x for line deletion for asynchronous remote devices.
Delete a character	To delete a character, depress the Backspace key. For terminals without a Backspace key, simultaneously depress the CTRL and character h keys.
End an input line	To process an input line, depress the carriage return (CR) key.
Communicate with MTM	To communicate with MTM while an interactive task is executing or when a BUILD command is active, depress the Break key and enter a command.

1.7.2.1 Using the Break Key

If the data request prompt (>) or a BUILD request prompt (B>) is displayed and the user wishes to communicate with MTM, depress the Break key and the system is ready to accept a command.

If input or output to the terminal is in progress, the Break key interrupts the process. For example, if the DISPLAY or EXAMINE command was entered and the output is in progress, depressing the Break key halts the output in progress. The system is then ready to accept a command.

If a CSS is currently running, the Break key interrupts the execution of the CSS. The system is then ready to accept a command. Once the command has executed, the CSS will resume operation unless the entered command affects the status of the CSS.

1.7.3 Command Conventions

Commands are accepted one line at a time. Multiple commands can appear on the same line, but each must be separated by a semicolon. Multiple commands are executed sequentially. If an error is encountered in a multiple command line that was entered from a terminal, the commands following the command in error are ignored by MTM. For a command line entered from a CSS, the commands on the command line are skipped until a $TERMJOB is found. A character string preceded by an asterisk in column 1 is a comment.

1.7.4 File Conventions

A file is a collection of data stored on a direct access storage device. MTM provides terminal users with the capability of creating and editing files in an interactive manner. Once created, files remain on the system until they are deleted by the owner. However, during the life of a file, ownership can change, based on the needs of an installation or project. File ownership is established and maintained by MTM via an account number mechanism.

1.7.4.1 Private Account Numbers

During the signon procedure a terminal user must supply a private account number in addition to the correct password. Whenever a terminal user allocates a file during an MTM session, the MTM system automatically associates the file with the terminal user's account number. A file associated with the terminal user's account number is referred to as a private file.

The owner of a private file has unrestricted access to that file and can update, execute, access, or delete it as required. Furthermore, no other terminal user except users with the correct privilege (privileged user) can gain access to another user's private files. However, to supply greater flexibility for file sharing, MTM supports the concept of group files.

1.7.4.2 Group Account Numbers

Authorized MTM terminal users are assigned both a private account number and a group account number within the AUF. Unlike the private account number, a terminal user is not required to submit the group account number during the signon procedure. In fact, a terminal user does not need to know the group account number. The group account number will generally be the private account number of a different authorized terminal user. By using the RENAME command and supplying the letter 'G' in the account field, a terminal user can change a private file to a group file.

As an illustration of the use of group files within an installation, consider a normal development activity consisting of two or more members working under a project leader's control. During the early development phase, each member would probably work alone, using private files. However, during the project integration phase, the majority of the private files would be switched to the project leader's private account number, which was defined as the group account for the individual members.

Once a private file has been switched to a group file, the original private owner no longer possesses unrestricted file manipulation capability. Instead, the file can be read or executed by the original owner and any other terminal user with the same group number. Updating or deleting the file can now be performed by any terminal user who signs on with the group account number.

Although the use of group files provides a somewhat flexible file sharing capability, it does not address the problem of universal sharing. For this purpose, MTM supports the concept of system files.

1.7.4.3 System Account Numbers

In a way similar to switching a private file to a group file, a terminal user can supply the letter 'S' in the file account field instead of the letter 'G'. The letter 'S' indicates that this private file is now considered a system file. System files have an account number of 0. They can be read or loaded by any authorized MTM terminal user. However, updating or deleting a system file can be performed only by the system operator.

Within an MTM environment, the system operator is viewed as more privileged than terminal users with respect to file ownership. The system operator can allocate files on any account in the system and can also change the account number of any file in the system to any other account number. Similar to a terminal user, the system operator uses the RENAME command to change file ownership.

1.7.4.4 File Descriptors (fds)

File descriptors are required with some commands. A file descriptor for MTM generally includes four fields:

- Disk volume name or device name
- Filename
- File extension
- File class

Format:

$$\left[\left\{\begin{array}{c}\text{voln:}\\ \text{user voln:}\end{array}\right\}\right] \text{filename}\left[.\left[\text{ext}\right]\right] \left[/\left\{\begin{array}{c}\text{P}\\ \text{G}\\ \text{S}\\ \text{n}\end{array}\right\}\right]$$

Parameters:

voln:	is the name of the disk volume on which the file resides, or the name of a device. Voln can be from one to four characters. The first character must be alphabetic and the remaining alphanumeric. This parameter need not be specified. If this parameter is not specified, the default user volume is used. When voln is not specified, the colon separating voln and filename must not be entered. Where voln refers to a device name, a colon must follow the device name, and neither the filename nor the extension is entered.
filename	is the name of a file. A filename consists of from one to eight alphanumeric characters, the first of which must be alphabetic.
.ext	is a 1- to 3-character alphanumeric string preceded by a period specifying the extension to a filename. If the period (.) and extension are omitted, a default extension is appended to the filename if appropriate for a particular command, otherwise, it remains blank. If the period is specified and the extension is omitted, the default is blanks.
P	indicates a private file. A private file has the same account number as the terminal user's current private account number. All of the facilities for file manipulation are available to the owner of this file. No other user has access to this file unless the user has certain standard file access privileges (privileged user) or, the file is also a group file. That is, the user's private account number is the same as some other user's group account number. P is the default value if neither P, G, nor S is indicated in the command.

G	indicates a group file. A group file, (which may also be some other user's private file), is accessible to members of that group for read only purposes. The group file account number in the AUF indicates to the system which users can access this group file.
S	indicates a system file. A system file has account number 0. A terminal user can only read a system file.
n	privileged users that have the privilege to specify account numbers instead of account class designators (P, G, and S) can do so for some commands such as ASSIGN, LOAD, RENAME, and CSS calls. Access is limited to SRO if n is not the user's private account.

Examples:

PACK:FRED.TSK	is a private file FRED.TSK on volume PACK.
FRED.TSK	is the same file as in the previous example, if PACK is the default user volume (private file).
ABC:FOO/G	is a group file with filename FOO with default extension, on volume ABC.
CARD:	is a device name.
A:B.C/G	is a group file B, with extension C on volume A.
TEXT.FIL/87	is a file on the default user volume in account 87.

CHAPTER 2
MULTI-TERMINAL MONITOR (MTM) USER COMMANDS

2.1 INTRODUCTION

The following steps comprise a basic MTM terminal session:

SIGNON MAR,118,SWDOC	Identify yourself to MTM by signing on to the system. Enter your userid, account number, and a valid password.
V M300	Establish the volume you will be working on by entering the VOLUME command and a valid volume name.
LOAD EDIT32	Load the editor task into memory by entering LOAD and the task name.
START . . .	Initiate execution of the task by entering the START command.
S FILE1	Save all data appended to your file by entering the SAVE command.
END	Terminate execution of the task by entering END.
SIGNOF	End the terminal session by signing off.

2.2 ALLOCATE COMMAND

The ALLOCATE command creates a direct access file or a communications line control block for a buffered terminal manager.

Format:

```
              ⎧ CONTIGUOUS,fsize [,{keys|0000}]
              ⎪ EC [/[{bsize|64}]] [/[{isize|3}]] [,[{keys|0000}]]
ALLOCATE fd,  ⎨ INDEX [,[{lrecl|126}]] [/[{bsize|1}]] [/[{isize|1}]] [,[{keys|0000}]]
              ⎪ NB [,[{lrecl|126}]] [/[{bsize|64}]] [/[{isize|3}]] [,[{keys|0000}]]
              ⎩ ITAM [,[{lrecl|126}]] [/[{bsize|1}]] [,[{keys|0000}]]
```

Parameters:

fd	is the file descriptor of the device or file to be allocated.
CONTIGUOUS	specifies that the file type to be allocated is contiguous.
fsize	is a decimal number indicating file size which is required for contiguous files. It specifies the total allocation size in 256-byte sectors. This size may be any value up to the number of contiguous free sectors existing on the specified volume at the time the command is entered.

keys	specifies the write and read protection keys for the file. These keys are in the form of a hexadecimal halfword, the left byte of which signifies the write key and the right byte, the read key. If this parameter is omitted, both keys default to 0.
EC	specifies that the file type to be allocated is extendable contiguous.
bsize	is a decimal number specifying the number of 256-byte sectors contained in a physical block to be used for buffering. This parameter cannot exceed the maximum block size established at sysgen time. If bsize is omitted, the default value is one sector for indexed files and 64 sectors for extendable contiguous (EC) and nonbuffered (NB) indexed files. When the file type is ITAM, bsize is the buffer size in bytes.
isize	is a decimal number specifying the indexed block size. If isize is omitted, the default value is one sector for indexed files and three sectors for EC and NB files. Like bsize, isize cannot exceed the maximum block size established at sysgen time.
INDEX	specifies that the file type to be allocated is indexed.
lrecl	is a decimal number specifying the logical record length of an indexed file, nonbuffered indexed file, or ITAM device. It cannot exceed 65,535 bytes. Its default is 126 bytes. It may optionally be followed by a slash (/) which delimits lrecl from bsize. For NB files, this number must be even.
NB	specifies that the file type to be allocated is nonbuffered indexed.
ITAM	specifies that the device to be allocated is a communications device.

Functional Details:

The MTM user can only allocate files in their private account. To assign an indexed file, sufficient room must exist in system space for two buffers, each of the stated size. Therefore, if bsize or isize is very large, the file might not be assignable in some situations. At sysgen time, a maximum block size parameter is established in the system, and bsize cannot exceed this constant.

To assign an EC or NB file, sufficient room must exist in system space to contain only the index block of the stated size. The data blocks for EC and NB files are not buffered in system space and thus are not constrained to the sysgened block size.

The ALLOCATE command can be entered in command mode, task loaded mode, and task executing mode.

Examples:

AL JANE.TSK,CO,64	Allocates, on the default user volume, a contiguous file named JANE.TSK whose total length is 64 sectors (16kb) with protection keys of 0.
AL M300:AJM.BLK,IN,132/4	Allocates, on volume M300, an indexed file named AJM.BLK with logical record length of 132 bytes, data block size of four sectors, and default isize of one sector. The protection keys default to 0. When this file is assigned, the system must have 2.25kb of available system space for buffers.
AL THISFILE,IN,256/4/2	Allocates, on the default user volume, an indexed file named THISFILE (blank extension) with a logical record length of 256 bytes, a data block size of four sectors, an index block size of two sectors, and protection keys of 0.
AL VOL1:AJM.OBJ,IN,126	Allocates, on volume VOL1, an indexed file named AJM.OBJ whose logical record length is 126 bytes. The buffer size and indexed block size default to one sector and the protection keys default to 0.

AL VO1:AJM.OBJ,IN,126//3	Allocates, on volume VO1, an indexed file named AJM.OBJ with logical record length of 126 bytes. The data block size defaults to one sector, the index block size is three sectors, and the protection keys default to 0.
AL SYS:XFILE.DTA,EC	allocates on volume SYS an extendable contiguous file named XFILE.DTA with default data block size of 64 and index block size of 3 sectors. The file initially contains no records, and has a record length of one sector (same as a contiguous file).
AL YFILE.DAT,NB,240/250/5	allocates on the default volume a nonbuffered indexed file named YFILE.DAT with logical record length of 240 bytes, data block size of 250 sectors, and index block size of 5 sectors. The file initially contains no records.

2.3 ASSIGN COMMAND

The ASSIGN command assigns a device, file, or communications device to one of a task's logical units.

Format:

```
                  ┌ ┌ ┌access privileges┐ ┐                  ┌ ┌SVC15 ┐ ┐ ┐
                  │ │ │SRW              │ │  ┌ ┌keys┐ ┐     │ │SVCF  │ │ │
                  │ │ │SREW             │ │  │ │    │ │     │ │VFC   │ │ │
ASSIGN lu,fd      │,│ │SRO              │ │  │,│    │ │     │,│HI    │ │ │
                  │ │ │ERW              │ │  │ │0000│ │     │ │LOW   │ │ │
                  │ └ └                 ┘ ┘  └ └    ┘ ┘     └ └MEDIUM┘ ┘ │
                  └                                                      ┘
```

Parameters:

lu	is a decimal number specifying the logical unit number to which a device or file is to be assigned.
fd	is the file descriptor of the device or file to be assigned.
access privileges	are the desired access privileges. The default access privileges are:

- SRW for contiguous files
- SREW for indexed, nonbuffered indexed, and extendable contiguous files
- SRO for any files that are not the users' private files
- ERW for devices (except the users' console. This has SRW.)

keys	signifies the read/write protection keys of the file or device to be assigned.

SVC15 SVCF	signifies that the specified device is to be assigned for SVC 15 access. SVCF is the hexadecimal equivalent of SVC15 and can also be specified. This option pertains to communications devices only. If SVC 15 access is specified, neither vertical forms control nor tape density can be specified.
VFC	specifies the use of vertical forms control for the assigned lu. If this parameter is specified, SVC15 access or tape density selection cannot be specified. If this parameter is omitted, there is no vertical forms control for the device assigned to the specified lu (unless the task was linked with the VFC option).
HI	indicates that the assigned magnetic tape will operate at the GCR density rate of 6250 bpi.
LOW	indicates that the assigned magnetic tape will operate at the NRZI density rate of 800 bpi.
MEDIUM	indicates that the assigned magnetic tape will operate at the PE density rate of 1600 bpi.

Functional Details:

If the access privileges and keys parameters are omitted and VFC, SVC15, HI, LOW, or MEDIUM are specified, the positional commas belonging to the omitted parameters can be omitted.

If the access privileges and VFC, SVC15, HI, LOW, or MEDIUM parameters are specified and the keys parameter is omitted, the positional comma belonging to the keys parameter can be omitted.

Access privileges can be one of the following:

SRO	sharable read-only
ERO	exclusive read-only
SWO	sharable write-only
EWO	exclusive write-only
SRW	sharable read/write
SREW	sharable read, exclusive write
ERSW	exclusive read, sharable write
ERW	exclusive read/write

If the file is not in the user's private account, only the SRO access privilege is valid.

When the SVC15 option is specified, only SRW, SREW, ERSW, and ERW access privileges are accepted.

The DISPLAY LU command can be used to determine the current access privileges of all assigned units.

The ASSIGN command is rejected if the requested access privilege cannot be granted.

When a task assigns a file, it might want to prevent other tasks from accessing that file while it is being used. For this reason, the user can ask for exclusive access privileges, either for read or for write, at assignment time. This is called dynamic protection because it is only in effect while the file remains assigned.

A file cannot be assigned with a requested access privilege if it is incompatible with some other existing assignment to that file. A request to open a file for exclusive write-only is compatible with an existing assignment for SRO or ERO, but is incompatible with any existing assignment for other access privileges. Table 2-1 illustrates compatibilities and incompatibilities between access privileges.

TABLE 2-1 ACCESS PRIVILEGE COMPATIBILITY

	ERSW	ERO	SRO	SRW	SWO	EWO	SREW	ERW
ERSW	-	-	-	-	*	-	-	-
ERO	-	-	-	-	*	*	-	-
SRO	-	-	*	*	*	*	*	-
SRW	-	-	*	*	*	-	-	-
SWO	*	*	*	*	*	-	-	-
EWO	-	*	*	-	-	-	-	-
SREW	-	-	*	-	-	-	-	-
ERW	-	-	-	-	-	-	-	-

LEGEND

*	compatible
-	incompatible

The keys format is a 4-digit hexadecimal number. The left two digits signify the write protection key and the right two digits, the read protection key. If omitted, the default is 0000. These keys are checked against the appropriate existing keys for the file or device. The command is rejected if the keys are invalid. The keys associated with a file are specified at file allocation time. They may be changed by a REPROTECT command or through an SVC 7 reprotect function call.

If the values of the keys are within the range X'01' to X'FE', the file or device cannot be assigned for read or write access unless the requesting task supplies the matching keys. If a key has a value of X'00', the file or device is unprotected for that access mode. Any key supplied is accepted as valid. If a key has a value of X'FF', the file is unconditionally protected for that access mode. It cannot be assigned for that access mode to any user task, regardless of the key supplied.

Examples:

WRITE KEY	READ KEY	MEANING
00	00	Completely unprotected
FF	FF	Unconditionally protected
07	00	Unprotected for read, conditionally protected for write (user must supply write key=X'07')
FF	A7	Unconditionally protected for write, conditionally protected for read
00	FF	Unprotected for write, unconditionally protected for read
27	32	Conditionally protected for both read and write

An assigned direct access file is positioned at the end of the file for access privileges SWO and EWO. It is positioned at the beginning of the file for all other access privileges. The command is rejected if the specified lu is already assigned. To reassign an lu for an active task, the lu must first be closed.

If one of the HI, LOW, or MEDIUM parameters is not chosen when assigning to a mag tape device, the standard default density is used. The default used is dependent upon the type of tape drive in use. Note that if this parameter is used to select the density of the assigned mag tape, SVC 15 or VFC access cannot be specified. The HI, LOW, and MEDIUM parameter options are positionally independent.

The ASSIGN command can be entered in task loaded mode.

Examples:

`AS 2,FILE.DAT,EWO,99AA`	Assigns a disk file to lu2. The EWO access privilege causes the file to be positioned at the end. It is conditionally protected with write and read keys of 99AA. New records are appended.
`AS 2,TEST.JOB,VFC`	Assigns a disk file to lu2. Vertical forms control is in use. Access privileges and keys parameters are omitted along with their respective commas.
`AS 2,TEST.JOB,,,VFC`	Assigns a disk file to lu2. Vertical forms control is in use. Access privileges and keys parameters are omitted but positional commas are specified.
`AS 2,TEST.JOB,,VFC`	Assigns a disk file to lu2. Vertical forms control is in use. The positional comma belonging to the omitted access privileges parameter must be specified.
`AS 2,TEST.JOB,SRO,VFC`	Assigns a disk file to lu2. Vertical forms control is in effect. The keys parameter, along with the positional comma, is omitted. The privilege is shared read only.
`AS 2,MAG1:,LOW`	Assigns a mag tape drive to lu2. The LOW parameter indicates that the drive will operate at the NRZI density rate of 800 bpi.
`AS 2,MAG1:,SRW,MEDIUM`	Assigns a mag tape drive to lu2. The MEDIUM parameter indicates that the drive will operate at the Perkin-Elmer density rate of 1600 bpi.
`AS 2,MAG1:,,,HI`	Assigns a mag tape drive to lu2. The HI parameter indicates that the drive will operate at the GCR density rate of 6250 bpi. Access privileges and keys parameters are omitted, but positional commas are specified.

Invalid Examples:

AS 2,TEST.JOB,OOFF,VFC	Invalid assignment because the positional comma belonging to the omitted access privileges parameter must be specified.
AS 2,TEST.JOB,SRO,VFC,SVC15	Invalid assignment because vertical forms control and SVC 15 access are mutually exclusive and cannot be specified in the same assignment.
AS 2,MAG1:,SRW,LOW,SVCF	Invalid assignment because tape density and SVCF access are mutually exclusive and cannot be specified in the same ASSIGN command.

2.4 BFILE COMMAND

The BFILE command backspaces to the preceding filemark on magnetic tapes, cassettes, and direct access files.

Format:

```
    BFILE [fd,] lu
```

Parameters:

fd	is the file descriptor of the device or file to be backspaced to a filemark.
lu	is the lu to which the file is assigned. If lu is specified without fd, the operation is performed on the lu regardless of what is assigned to it.

Functional Details:

The BFILE command can be entered in task loaded mode.

Examples:

BF 1	Causes the device or file assigned to lu1 to backspace one filemark.
BF M300:AJM.OBJ,4	Causes file AJM.OBJ, that is assigned to lu4 on volume M300:, to backspace one filemark.

2.5 BIAS COMMAND

The BIAS command sets a base address for the EXAMINE and MODIFY commands.

Format:

$$\text{BIAS}\begin{Bmatrix}\text{address}\\ *\end{Bmatrix}$$

Parameters:

address	is a hexadecimal bias to be added to the address given in any subsequent EXAMINE or MODIFY command. For a u-task, the address must be a valid address that exists for the u-task. For an e-task, the address can be any valid address in the system. The addresses must be aligned on a halfword boundary. If address is omitted, it is assumed to be the beginning of the task.
*	sets bias to 0 for a u-task and to the physical load address for an e-task.

Functional Details:

A BIAS command overrides all previous BIAS commands. The user should enter a BIAS command if the current value is unknown.

The BIAS command can be entered in task loaded mode and task executing mode.

Example:

```
    BI 100                   Sets bias to 100
```

2.6 BREAK COMMAND

The BREAK command returns a break status (X'8200') to a task with an outstanding I/O on the MTM terminal.

Format:

```
    BREAK
```

Functional Details:

The BREAK command can be entered in task executing mode.

2.7 BRECORD COMMAND

The BRECORD command backspaces to the preceding record on magnetic tapes, cassettes, and direct access files.

Format:

```
BRECORD [fd,] lu
```

Parameters:

fd	is the file descriptor of the device or file to be backspaced one record.
lu	is the lu to which the file is assigned. If lu is specified without fd, the operation is performed on the lu regardless of what is assigned to it.

Functional Details:

The BRECORD command can be entered in task loaded mode.

Examples:

BR 1	Causes the device or file assigned to lu1 to backspace one record.
BR M300:AJM.OBJ,4	Causes the file AJM.OBJ, assigned to lu4 on volume M300, to backspace one record.

2.8 BUILD AND ENDB COMMANDS

The BUILD and ENDB commands copy data from the command input device to the fd specified in the BUILD command.

Format:

$$\underline{\text{BU}}\text{ILD} \begin{Bmatrix} \text{fd} \\ \text{lu} \end{Bmatrix} [,\text{APPEND}]$$

```
  .
  .
  .
ENDB
```

Parameters:

fd	is the file descriptor of the device or file to which data is copied. If fd does not contain an extension, .CSS is used as a default. If a blank extension is desired, the period following the filename must be typed. If fd refers to a direct access file, an indexed file by that name is allocated with a logical record length equal to the command buffer length established at sysgen time, a blocksize of 1, and keys of 0000. If the specified fd already exists, that fd is deleted and a new fd is allocated.
lu	is the lu to which data is to be copied. A temporary file is allocated and the BUILD data is copied to it. When the ENDB is encountered, the temporary file is assigned to the specified lu of the loaded task. This form of the BUILD command is only valid when a task is loaded.
APPEND	allows the user to append data to an existing fd. If the fd does not exist, it is allocated.

Functional Details:

Lines entered from the terminal after the BUILD command are treated as data, and are copied to the specified device or file until an ENDB command is encountered. ENDB may be followed by other commands in the command line. Data following the ENDB command is treated as a command. If any data follows the BUILD command on the same line, it is treated as a comment and no action is taken. The BUILD command can be entered from the terminal only if a CSS is not active. It can be entered in command, task loaded, and task executing modes.

Example:

```
BUILD ASSN
AS 1, CR:
AS 2, OUT.OBJ
AS 3, PR:
AS 5, CON:
ENDB
```

2.9 CANCEL COMMAND

The CANCEL command terminates a task with an end of task code of 255.

Format:

```
    CANCEL
```

Functional Details:

The normal response to this command is:

```
    Signon name    END OF TASK CODE=255   CPUTIME=utime/ostime
```

The CANCEL command can be entered in task loaded mode and task executing mode.

2.10 CLOSE COMMAND

The CLOSE command closes (unassigns) one or more files or devices assigned to the currently selected task's logical units.

Format:

$$\underline{C}\text{LOSE} \left\{ \begin{array}{l} lu_1 \; \left[, lu_2, \ldots, lu_n\right] \\ \text{ALL} \end{array} \right\}$$

Parameters:

lu	decimal numbers signifying the logical units to be closed.
ALL	specifies that all logical units of the task are to be closed.

Functional Details:

Closing an unassigned lu does not produce an error message. A CLOSE command can only be entered if the task is dormant or paused.

The CLOSE command can be entered in task loaded mode.

Examples:

CL 1,3,5	Closes logical units 1, 3, and 5 of the task.
CLOSE A	Closes all logical units of the task.

2.11 CONTINUE COMMAND

The CONTINUE command causes a paused task to resume operation.

Format:

```
    CONTINUE [address]
```

Parameter:

address	is a hexadecimal number that specifies where the task is to resume operation. If this parameter is not specified or is 0, the task resumes at the instruction following the pause.

Functional Details:

The CONTINUE command can be entered in task loaded mode. Executing this command causes the terminal mode to be switched from task loaded mode to task executing mode.

2.12 DELETE COMMAND

The DELETE command deletes a direct access file.

Format:

```
DELETE fd₁ [,fd₂ ,...,fdₙ]
```

Parameter:

fd	identifies the file(s) to be deleted.

Functional Details:

The file being deleted must not be currently assigned to an lu of any task. A file can be deleted only if its write and read protection keys are 0 (X'0000'). If the keys are nonzero, they can be changed using the REPROTECT command. Only private files can be deleted.

The DELETE command can be entered in command mode, task loaded mode, and task executing mode.

2.13 DISPLAY COMMAND

The DISPLAY command is used to display new global or new internal variables currently defined by the user. This command will not display local variables or global variables.

Format:

$$\text{\underline{D}ISPLAY} \left\{\begin{matrix}\text{\underline{G}VARIABLE}\\ \text{\underline{I}VARIABLE}\end{matrix}\right\} \left[\,,\left[\left\{\begin{matrix}n_1/n_2\\ n\\ \text{ALL}\end{matrix}\right\}\right],\left[\left\{\begin{matrix}\text{fd}\\ \text{user console}\end{matrix}\right\}\right]\right]$$

Parameters:

GVARIABLE	indicates that the variables to be displayed are new global variables.
IVARIABLE	indicates that the variables to be displayed are new internal variables.
n_1/n_2	specifies that all variables (of the type selected via the preceding parameter) between the range n_1 to n_2 be displayed. Where n is a decimal number between 1 and the maximum value allowed at MTM sysgen for the variable type selected.
n	is the decimal number of a specific variable. n must be between 1 and the maximum value allowed at MTM sysgen for the variable type selected.
ALL	specifies that all new global or new internal variables be displayed. This is the default if no specific variable numbers are entered.
fd	is a file descriptor of a file or device to which the display is to be output. The default for this parameter is the users console.

Functional Details:

The DISPLAY command can be used in command mode, task loaded mode, and task executing mode.

The current value of each variable is displayed in the DISPLAY command display.

Examples:

Example 1 illustrates a means of displaying all new global variables currently defined by the user.

```
*DISPLAY GVARIABLE

GV#     NAME....VALUE.............................................
G01     SOURCE  TEST.FTN/P
G03     LISTDEV SCRT:TEST.LST/P
G04             BATCH OPTIM XREF
```

Example 2 illustrates a means of displaying information about new global variable 3.

```
*DISPLAY GVARIABLE, 3

GV#     NAME....VALUE.............................................
G03     LISTDEV SCRT:TEST.LST/P
```

Example 3 illustrates a means of displaying all new global variables between 2 and 5.

```
*DISPLAY GVARIABLE, 2/5

GV#     NAME....VALUE.............................................
G03     LISTDEV SCRT:TEST.LST/P
G04             BATCH OPTIM XREF
```

2.14 DISPLAY ACCOUNTING COMMAND

The DISPLAY ACCOUNTING command displays accounting data collected for a currently running or previously run task.

Format:

$$\underline{\text{D}}\text{ISPLAY }\underline{\text{A}}\text{CCOUNTING}\left[,\left\{\begin{matrix}\text{fd}\\ \text{user console}\end{matrix}\right\}\right]$$

Parameter:

fd	is the file descriptor to which the accounting information is displayed. The user console is the default.

Functional Details:

The DISPLAY ACCOUNTING command displays this information:

```
USER TIME hh:mm:ss.ms
SVC  TIME hh:mm:ss.ms
WAIT TIME hh:mm:ss.ms
ROLL TIME hh:mm:ss.ms
I/O       n
ROLLS     n
```

The DISPLAY ACCOUNTING command can be entered in command mode, (providing at least one task has been run during the current terminal session), task loaded mode, and task executing mode.

2.15 DISPLAY DEVICES COMMAND

The DISPLAY DEVICES command displays to the specified fd the physical address, keys, online/offline state, and the volume name (for online direct access devices) of all devices in the system.

Format:

$$\text{\underline{D}ISPLAY \underline{D}EVICES}\left[,\begin{Bmatrix}\text{fd}\\ \text{user console}\end{Bmatrix}\right]$$

Parameter:

fd	is the file descriptor specifying the file or device to which the display is routed. If fd is omitted, the default is the user console.

Functional Details:

The DISPLAY DEVICES command can be entered in command mode, task loaded mode, and task executing mode.

Example:

```
D D

NAME  DN KEYS
NULL   0 0000                          D300  FC 0000   M300  CD
D301  DC 0000   M301  CD               D67A  EC 0000   M67A  CD
D67B  ED 0000   MTM   SYS  CD          D05A  C6 0000   OFF
D058  C7 0000   FIXD  CD               MAG2  95 0000
MAG3  C5 0000                          MAG4  D5 0000
CON    2 0000                          CR     4 0000
PRT   63 0000                          PR     0 0000   SPOL
PR1    0 0000   SPOL                   CT34  34 0000
CT36  36 0000                          CT3C  3C 0000
CT42  42 0000                          CT46  46 0000
CT4C  4C 0000                          CT72  72 0000
CT74  74 0000                          CT7A  7A 0000
CT7C  7C 0000                          IT7E  7E 0000  ITAM
DI18  18 0000  ITAM                    BI18  18 0000
BQLA  B8 0000  ITAM                    BQ2A  B8 0000  ITAM
BQ3A  B8 0000  ITAM                    BQPA  B8 0000  ITAM
BQLB  BC 0000  ITAM                    BQ2B  BC 0000  ITAM
BQ3B  BC 0000  ITAM                    BQPB  BC 0000  ITAM
IRDR:********.***
```

In the DISPLAY DEVICES output the screen or page is divided in half in order to display more devices per page (or screen). The definition of the columns is applicable to either half of the display. Columns 1, 2, and 3 contain the device name, device number (address), and keys, respectively. Column 4 is only defined for pseudo-print (spool), ITAM (communications), and direct access devices. The characters SPOL specify that the devices are pseudo-print devices used in spooling.

For direct access devices, column 4 contains the characters OFF to indicate that the device is offline. If online, the volume name is output in column 4. For write-protected disks, column 5 contains the characters PROT. For MTM users, if the disk is write-protected, column 5 contains the characters SYS. If the disk is restricted, column 5 contains the characters RES. If the secondary directory option is enabled, the last column contains the characters CD.

Pseudo devices created by the SVC intercept facility are displayed as a file descriptor with asterisks filling the filename and extension fields. As an example, all SPL/32 spooler pseudo devices are displayed in this manner.

2.16 DISPLAY DFLOAT COMMAND

The DISPLAY DFLOAT command displays to the specified fd the contents of the double precision floating point registers associated with the loaded task.

Format:

$$\text{DISPLAY DFLOAT}\left[,\begin{Bmatrix}\text{fd}\\ \text{user console}\end{Bmatrix}\right]$$

Parameter:

fd	is the file descriptor specifying the file or device to which the contents of the double precision floating point registers associated with a user-specified task are displayed. If fd is omitted, the default is the user console.

Functional Details:

The user-specified task should have been built with the DFLOAT option at Link time.

The DISPLAY DFLOAT command can be entered in task loaded and task executing mode.

Example:

```
D DFL
0,2  00000000  00000000    00000000  00000000
4,6  00000000  00000000    00000000  00000000
8,A  00000000  00000000    00000000  00000000
C,E  00000000  00000000    00000000  00000000
```

2.17 DISPLAY FILES COMMAND

The DISPLAY FILES command permits information from the directory of one or more direct access files to be output to a specified fd.

Format:

```
                  ┌(        :         )┐
DISPLAY FILES ,   │{      voln:       }│ [filename] [.[ext]]
                  └(default user vol  )┘
                 ┌ ┌([illegible])┐                          ┐
                 │ │ S │                                    │
                 │ │ G │     ┌(     fd      )┐              │
                 │/│{  }│   ,│{             }│              │
                 │ │ N │     └(user console )┘              │
                 │ │ O │                                    │
                 └ └ L ┘                                    ┘
```

NOTE

Please see Functional Details for variations on the DISPLAY FILES command syntax.

Parameters:

:	specifies that all files with the user account number be displayed regardless of what volume they reside on. Entering the colon with part of a filename limits the file search to filenames with the specified characters.
voln:	is a 1- to 4-character name of a disk volume. The first character must be alphabetic, the remaining alphanumeric. If voln is omitted, the default is the user volume.

filename	is a 1- to 8-character name of a file. The first character must be alphabetic, the remaining, alphanumeric.
ext	is a 1- to 3-character extension to the filename.
P	indicates that information is requested for a private file.
S	indicates that information is requested on a system file; default is private files only.
G	indicates that information is requested for a group file; default is private files only.
N	indicates that information is requested for private and group files.
O	indicates that information is requested for group and system files.
L	indicates that information is requested for private and system files.
fd	is the file descriptor specifying the file or the device to which the display is output. If fd is omitted, the default is the user console.

Functional Details:

A hyphen (-) in the command format requests that all files starting with the characters preceding the - or following the - are displayed, subject to any restrictions specified in the extension, account number, and fd fields. For example:

CAL32-	displays all files whose first five characters are CAL32.
CAL32.-	displays all files named CAL32 with any extension.
-.MTM	displays all files with the the extension MTM.
CH-.043	displays all files beginning with CH, with an extension of 043.

The character * requests that all files with matching characters in the same position(s) as those entered are displayed. For example:

CAL32***	displays all files between five and eight characters in length whose first five characters are CAL32.
CAL**CAL	displays all files, with a filename eight characters long, whose first three and last three characters are CAL.
****32.OBJ	displays all files with a filename containing six characters whose fifth and sixth characters are 32 and whose extension is .OBJ.

An asterisk in the account position indicates that all accounts are to be searched for a match. If the user is a privileged user, every account on the system is checked. If the user is a nonprivileged user, the P, G, and S accounts are checked.

The characters * and - can be combined in the command format, as described previously, to further delimit files displayed. For example:

CAL**1-	displays all files whose first three characters are CAL, and whose sixth character is 1.
****32.O-	displays all files, eight characters long, whose last two characters are 32 and whose extension begins with an O.

A colon entered with part of a filename and a dash displays all filenames with the user account number starting with the specified characters, regardless of what volume they reside on:

```
D F, :JM-
```

A colon entered with a specified extension displays all files under the user account number with the specified extension, regardless of what volume they reside on:

```
D F,:.JM
```

An example of the display produced by the DISPLAY FILES command from a privileged user is:

```
    M300:-.-
```

```
VOLUME= M300
 FILENAME.....        TY DBS/IBS RECL. RECORDS CREATED....... LAST WRITTEN.. KEYS
 SYSEDIT .CMD/00205 IN   1/1     80       1 11/10/82 22:30 11/10/82 22:30 0000
 TEST    .CSS/00205 IN   1/1    132       2 11/15/82 11:30 11/15/82 11:30 0000
 CONTIG  .   /00205 CO                   35 11/15/82 11:35 11/15/82 11:35 0000
 IN      .   /00205 IN  10/3     50       0 11/15/82 11:35 11/15/82 11:35 0000
```

An example of the same DISPLAY FILES command from a nonprivileged user is:

```
    D F, M300:-.-/P
```

```
VOLUME= M300
 FILENAME......     TY DBS/IBS RECL. RECORDS CREATED....... LAST WRITTEN.. KEYS
 SYSEDIT .CMD/P     IN   1/1     80       1 11/10/82 22:30 11/10/82 22:30 0000
 TEST    .CSS/P     IN   1/1    132       2 11/15/82 11:30 11/15/82 11:30 0000
 CONTIG  .   /P     CO                   35 11/15/82 11:35 11/15/82 11:35 0000
 IN      .   /P     IN  10/3     50       0 11/15/82 11:35 11/15/82 11:35 0000
```

For contiguous files, TYPE (TY) is CO, and RECORDS is the size of the file in (decimal) sectors.

For indexed files, TYPE is IN, followed by the data and index blocking factors, RECL is the logical record length in (decimal) bytes, and RECORDS is the number of logical records (in decimal) in the file.

For nonbuffered indexed files, TYPE is NB, RECL is logical record length in (decimal) bytes, and RECORDS is the number of logical records (in decimal) in the file.

For extendable contiguous files, TYPE is EC, and RECORDS is the length of the file in sectors (i.e., the size of the file).

Spool and temporary files are named as *SPOOLFILE* and *TEMPFILE* respectively (unless the user has the privilege to see the actual filenames, in which case, the names are displayed).

The DISPLAY FILES command can be entered in command mode, task loaded mode, and task executing mode.

NOTE

If a DISPLAY FILES command is entered by a privileged user, the account number of each file is displayed. Nonprivileged MTM users see the account class (P, G, or S).

Examples:

D F	displays to the user terminal all files with the user's account number on the default user volume.
D F,CAL32.TSK/-	displays file CAL32.TSK in the private, group, and system accounts.
D F,-/-	displays all files in the private group and system accounts on the default user volume.
D F,,MAG1:	displays, to the device MAG1, all files with the user's account number on the default user volume.
D F,M300:	displays, to the user's terminal, all files with the user's account number on volume M300.
D F,M300:A-.TSK	displays all files on volume M300 with first character A and extension TSK in the user's account number.
D F,-.,PR1:	displays all files on the default user volume in the user's account number with blank extension, regardless of filename. The display is routed to device PR1:.
D F,CAL**1-.-	displays, to the user's terminal, all files that start with CAL, contain the character 1 in the sixth position, have any extension and are in the user's account number.

D F,M-:TASK.5*	displays to the user's terminal the files named TASK that have one or two character extensions starting with the character 5. A separate display of these files is done for each online disk volume whose name starts with the letter M.
D F,-:TASK.-	displays to the user's terminal the files named TASK, with any extension. A separate display of these files is done for each online disk volume in the system.
D F,-:EDIT-/*	displays all files that start with the four characters EDIT, on all volumes, in all accounts, regardless of the extension. If the user is not privileged, only matching files in the private, group, and system accounts are displayed.
D F,-/N	displays all files in the user's private and group account on the default users volume.

2.18 DISPLAY FLOAT COMMAND

The DISPLAY FLOAT command displays to the specified fd the contents of the single precision floating point registers associated with the loaded task.

Format:

$$\underline{D}\text{ISPLAY }\underline{F}\text{LOAT}\left[,\left\{\begin{array}{c}\text{fd}\\ \text{user console}\end{array}\right\}\right]$$

Parameter:

fd	is an optional file descriptor specifying the file or device to which the display is output. If fd is omitted, the display is output to the user's terminal.

Functional Details:

The user-specified task must be built with the FLOAT option specified at Link time.

The DISPLAY FLOAT command can be entered in task loaded mode.

Example:

```
D FL
0,2  00000000   00000000
4,6  00000000   00000000
8,A  00000000   00000000
C,E  00000000   00000000
```

2.19 DISPLAY LU COMMAND

The DISPLAY LU command displays to the specified fd all assigned logical units of the loaded task.

Format:

```
                 ┌ ┌            ┐ ┐
DISPLAY LU       │,│     fd      │ │
                 └ │user console│ ┘
                   └            ┘
```

Parameter:

fd	is an optional file descriptor specifying the file or device to which the assigned logical units are to be displayed. If fd is omitted, the default is the user console.

Functional Details:

The lu number, file or device name, current access privileges, current record number, and percentage thru file are displayed. The current record number and percentage thru file are displayed only for files.

```
LU   FILE/DEVICE                        RECORD   THRU
 1   M67A:RADPROC.CSS/000,SRO               30   15.0%
 3   CON:,SRW
 5   CON:,SRW
 6   CON:,SRW


 1   M67A:RADPROC.CSS/000,SRO              200  100.0%
 3   CON:,SRW
 4   M67A:&2614586.001/000,SREW              1  100.0%
 5   CON:,SRW
 6   CON:,SRW
```

The DISPLAY LU command can be entered in task loaded mode and task executing mode.

Example:

DISP LU,PR:	Displays assigned logical units to the printer device (PR:).

2.20 DISPLAY PARAMETERS COMMAND

The DISPLAY PARAMETERS command displays the parameters of the loaded task.

Format:

$$\text{DISPLAY PARAMETERS}\left[,\left\{\begin{array}{c}\text{fd}\\ \text{user console}\end{array}\right\}\right]$$

Parameter:

fd	is an optional file descriptor specifying the file or device to which the display is output. If fd is omitted, the default is the user console.

Functional Details:

Table 2-2 lists the field addresses and data displayed when the DISPLAY PARAMETERS command is entered.

TABLE 2-2 DISPLAY PARAMETERS COMMAND FIELDS

FIELD	VALUE	MEANING
TASK	xxxxxxxx	Task name, also user signon name
CTSW	xxxxxxxx	Status portion of current TSW
CLOC	xxxxx	Current location
STAT	xxxxx	Task wait status
TOPT	xxxxxxx	Task options
USSP	xxxxx	Current used system space

TABLE 2-2 DISPLAY PARAMETERS COMMAND FIELDS
(Continued)

FIELD	VALUE	MEANING
MUSP	xxxxx	Maximum used system space
MXSP	xxxxx	Maximum allowed system space
CTOP	xxxxx	Task CTOP
UTOP	xxxxx	Task UTOP
UBOT	xxxxx	Task UBOT
SLOC	xxxxx	Task starting location
NLU	xxx	Number of logical units (decimal)
MPRI	xxx	Maximum priority (decimal)
SVOL	xxxx	Default volume ID

The addresses displayed as CTOP, UTOP, and UBOT, are not physical addresses, but addresses within the task's own program space. CLOC may be a program space address or a physical address in a system subroutine being executed on behalf of the task. NLU is given in decimal. SVOL is the ASCII default volume ID. The fields CTOP, UTOP, UBOT, and SLOC are described in detail in the OS/32 Application Level Programmer Reference Manual.

TOPT is given in hexadecimal. The definitions of task option bits are listed in Table 2-3.

TABLE 2-3 TASK OPTION BIT DEFINITIONS

BIT	MASK	MEANING
4	0800 0000	0 = Dynamic scheduling disabled 1 = Dynamic scheduling enabled
5	0400 0000	0 = Prompt disabled 1 = Prompt enabled
6	0200 0000	0 = I/O interpreted without VFC 1 = All I/O interpreted with VFC

TABLE 2-3 TASK OPTION BIT DEFINITIONS (Continued)

BIT	MASK	MEANING
7	0100 0000	0 = No extended SVC 1 parameter blocks used (excludes communications I/O) 1 = Extended SVC 1 parameter blocks used
8	0080 0000	0 = New TSW for task event service 1 = No new TSW for task event service
9	0040 0000	0 = Task event all registers saved 1 = Task event partial registers saved
10	0020 0000	0 = Task event no register saved 1 = Task event register saved
16	0000 8000	0 = U-task 1 = E-task
17	0000 4000	0 = AFPAUSE 1 = AFCONT
18	0000 2000	0 = NOFLOAT 1 = Single floating point
19	0000 1000	0 = NONRESIDENT 1 = RESIDENT
20	0000 0800	0 = SVC 6 control call 1 = Prevent SVC 6 control call
21	0000 0400	0 = SVC 6 communication call 1 = Prevent SVC 6 communication call
22	0000 0200	0 = SVCPAUSE 1 = SVCCONT
23	0000 0100	0 = NODFLOAT 1 = DFLOAT
24	0000 0080	0 = NOROLL 1 = ROLL
25	0000 0040	0 = No overlay 1 = Use overlay
26	0000 0020	0 = Accounting disabled 1 = Accounting enabled
27	0000 0010	0 = Task can issue intercept call 1 = Task cannot issue intercept call

TABLE 2-3 TASK OPTION BIT DEFINITIONS (Continued)

BIT	MASK	MEANING
28	0000 0008	0 = No account privileges 1 = File account privileges
29	0000 0004	0 = Bare disk assign not allowed 1 = Bare disk assign allowed
30	0000 0002	0 = Not universal 1 = Universal
31	0000 0001	0 = No keychecks 1 = Do keychecks

STAT is given in hexadecimal. The definitions of wait status bits are shown in Table 2-4.

TABLE 2-4 WAIT STATUS BIT DEFINITIONS

BIT	MASK	MEANING
15	0001 0000	Intercept wait
16	0000 8000	I/O wait
17	0000 4000	(Any) IOB/WAIT
18	0000 2000	Console wait (paused)
19	0000 1000	Load wait
20	0000 0800	Dormant
21	0000 0400	Trap wait
22	0000 0200	Time of day wait
23	0000 0100	Suspended
24	0000 0080	Interval wait
25	0000 0040	Terminal wait
26	0000 0020	Roll pending wait

TABLE 2-4 WAIT STATUS BIT DEFINITIONS (Continued)

BIT	MASK	MEANING
27	0000 0010	Intercept initialization (MTM)
28	0000 0008	Intercept termination (MTM)
29	0000 0004	System resource connection wait
30	0000 0002	Accounting wait

NOTE

Zero status indicates an active task.

CTSW is expressed in hexadecimal. For a definition of the status portion of the TSW, see the OS/32 Application Level Programmer Reference Manual.

The DISPLAY PARAMETERS command can be entered in task loaded mode and task executing mode.

Example:

The following is an example of the output generated in response to a DISPLAY PARAMETERS command:

```
*DISPLAY PARAMETERS


TASK    MTMUSER
CTSW    00001000
CLOC       F2B7C
STAT        2000
TOPT       10021
USSP        14F8
MUSP        2208
MXSP        3000
CTOP        24FE
UTOP        2370
UBOT           0
SLOC       F0000
NLU           15
MPRI         128
SVOL        M67A
```

2.21 DISPLAY REGISTERS COMMAND

The DISPLAY REGISTERS command displays to the specified fd the contents of the general purpose user registers associated with a loaded task.

Format:

$$\underline{D}\text{ISPLAY }\underline{R}\text{EGISTERS}\left[,\left\{\begin{matrix}\text{fd}\\ \text{user console}\end{matrix}\right\}\right]$$

Parameter:

fd	is the file descriptor to which the contents of the general purpose user registers are displayed. If fd is omitted, the display is output to the user console.

Functional Details:

The DISPLAY REGISTERS command can be entered in task loaded mode and task executing mode.

NOTE

The contents of each register will be 0 until the task has started.

Example:

```
D R
PSW  000077F0   0000E588
0-3  00000000   00000000   00000000   00004801
4-7  0000E83C   00000000   00000000   0000D2EA
8-B  0000E8CB   00000000   0000E848   00000028
C-F  0000E804   0000E9D0   0000E584   0000E05E
```

2.22 DISPLAY TIME COMMAND

The DISPLAY TIME command displays the current date and time to a specified fd.

Format:

$$\underline{\text{D}}\text{ISPLAY }\underline{\text{T}}\text{IME } \left[, \begin{Bmatrix} \text{fd} \\ \text{user console} \end{Bmatrix} \right]$$

Parameter:

fd	specifies the file or device to which the display is to be output. If fd is omitted, the default is the user console.

Functional Details:

The display has the following format:

```
mm/dd/yy    hh:mm:ss
```

or alternatively (by sysgen option):

```
dd/mm/yy    hh:mm:ss
```

The DISPLAY TIME command can be entered in command mode, task loaded mode, and task executing mode.

2.23 DISPLAY USERS COMMAND

The DISPLAY USERS command displays the userid, terminal device names, and the operating mode of all users currently signed on under MTM. Additionally, all active batch jobs are displayed.

Format:

$$\text{DISPLAY USERS}\left[,\begin{Bmatrix}\text{fd}\\ \underline{\text{user console}}\end{Bmatrix}\right]$$

Parameter:

fd	specifies the file or device to which the display is output. If fd is omitted, the default is the user console.

Functional Details:

This command can be entered in command mode, task loaded mode, and task executing mode.

Example:

```
D U
R-NULL:@$HASP00          BG-CT22:@MTM-MODE        NERD-NULL:@$STAT
LFS-CT26:>MTM-MODE       JON-CT32:@ECM-MODE       VAL-CT2A:>MTM-MODE
GRAY-CT2C:@MTM-MODE      LYNDA-NULL:@$STAT        BJM-CT30:@MTM-MODE
DAVE-CT3A:>MTM-MODE      BRI-CT3E:>MTM-MODE       JOB3-BATCH>MTM-MODE
```

>	- denotes nonprivileged MTM user
@	- denotes privileged MTM user
MTM-MODE	- standard MTM usage
ECM-MODE	- environmental control monitor mode
$	- foreground task mode and HASP mode
BATCH	- denotes an active batch job

2.24 ENABLE COMMAND

The ENABLE command allows the prompt or messages previously suppressed by the PREVENT command to be displayed on the user console.

Format:

```
        (MESSAGE  )
        |PROMPT   |
ENABLE <           >
        |ETM      |
        ($VARIABLE)
```

Parameters:

MESSAGE	allows other MTM users to send messages to the user terminal.
PROMPT	requests the system to print the hyphen (-) prompt in task executing mode. The hyphen (-) is the default prompt for task executing mode.
ETM	displays the end of task message.
$VARIABLE	enables variable processing of local and global variables on a per user basis.

Functional Details:

The ENABLE command does not affect operator messages.

Local and global variable support is included in the target sysgen option SGN.VAR.

2.25 EXAMINE COMMAND

The EXAMINE command examines the contents of a memory location in the loaded task.

Format:

$$\underline{\text{EXA}}\text{MINE address}_1 \left[\begin{Bmatrix} ,\text{n} \\ /\text{address}_2 \\ ,1 \end{Bmatrix}\right] \left[,\begin{Bmatrix} \text{fd} \\ \text{user console} \end{Bmatrix}\right]$$

Parameters:

address	indicates the starting and ending addresses in memory whose contents are to be displayed in hexadecimal. All addresses specified are rounded down to halfword boundaries by the system.
n	is a decimal number specifying the number of halfwords to be displayed. If n is omitted, one halfword is displayed.
fd	is the file descriptor specifying the file or device to which the contents of memory are displayed. If omitted, the default is the user console.

Functional Details:

Specifying only address_1 causes the contents of memory at that location (as modified by any previous BIAS command) to be displayed. Specifying address_1 and address_2 causes all data from the first to the second address to be displayed.

The EXAMINE command can be entered in task loaded mode and task executing mode.

Any memory that can be accessed by the loaded task can be examined with the EXAMINE command. For example, if a task uses a PURE segment that is mapped to segment register F, then examining addresses at F0000 or greater will display the contents of the PURE segment.

Example:

BI B100 EXA 100,10	Examines 10 halfwords starting at relative address 100 (absolute address B200) within the task.

2.26 FFILE COMMAND

The FFILE command forward spaces to the next filemark on magnetic tapes, cassettes, and direct access files.

Format:

```
    FFILE [fd,] lu
```

Parameters:

fd	is the file descriptor of the device or file, to be forward spaced one filemark.
lu	is the lu to which the file is assigned. If lu is specified without fd, the operation is performed on the lu regardless of what is assigned to it.

Functional Details:

The FFILE command can be entered in task loaded mode.

Examples:

FF 1	Causes the file or device assigned to lu1 to forward space one filemark.
FF M300:AJM.OBJ,4	Causes the file AJM.OBJ on volume M300 that is assigned to lu4, to forward space one filemark.

2.27 FRECORD COMMAND

The FRECORD command forward spaces one record on magnetic tapes, cassettes, and direct access files.

Format:

```
    FRECORD [fd,] lu
```

Parameters:

fd	is the file descriptor of the device or file to be forward spaced one record.
lu	is the lu to which the device or file is assigned. If lu is specified without fd, the operation is performed on the lu regardless of what is assigned to it.

Functional Details:

The FRECORD command can be entered in task loaded mode.

Examples:

FR 1	Causes the device or file assigned to lu1 to forward space one record.
FR M300:AJM.OBJ,4	Causes file M300:AJM.OBJ on volume M300 that is assigned to lu 4 to forward space one record.

2.28 HELP COMMAND

The HELP command displays information on MTM user and program development commands.

Format:

```
     ┌┌ mnemonic ┐┐
HELP ││          ││
     └└    *     ┘┘
```

Parameters:

mnemonic	is any valid MTM or program development command mnemonic.
*	causes a list of all MTM and program development commands to be displayed to the list device.

Functional Details:

The HELP command is implemented as a CSS procedure. When a mnemonic or command is entered, information on how to use that particular command is displayed to the list device. If parameters are omitted, information on how to use the HELP command is displayed to the list device.

Examples:

HELP LOG	Displays to the list device information on how to use the MTM LOG command.
HELP COMPILE	Displays to the list device information on how to use the program development command, COMPILE.
HELP	Displays to the list device information on how to use the HELP command.

Example:

```
HELP *
ADD              AL(LOCATE)       AS(SIGN)         BF(ILE)
BI(AS)           BREA(K)          BR(ECORD)        BU(ILD)
CAL              CA(NCEL)         CL(OSE)          COBOL
COMMAND          COMPILE          COMPLINK         CO(NTINUE)
DE(LETE)         D(ISPLAY)        EDIT             ENA(BLE)
ENDB             ENV              EXA(MINE)        EXEC
FF(ILE)          FILEDESC         FORT             FORTO
FORTZ            FR(ECORD)        HELP             INIT
LINK             LIST             L(OAD)           LOG
MACRO            ME(SSAGE)        MO(DIFY)         O(PTION)
PASCAL           PAS(SWORD)       P(AUSE)          PRE(VENT)
PRI(NT)          PUN(CH)          REL(EASE)        REMOVE
REN(AME)         REP(ROTECT)      REW(IND)         RPG
RUN              RW               RVOL(UME)        SEN(D)
SE(T)            SIGNOF(F)        S(IGNON)         SPOOLFILE
ST(ART)          T(ASK)           TE(MPFILE)       V(OLUME)
WF(ILE)          XAL(LOCATE)      XDE(LETE)        SUB(MIT)
INQ(UIRE)        PUR(GE)

For HELP on any of the above command mnemonics, type HELP
<command>
```

Example:

```
HELP PASSWORD
```

PASSWORD: The PASSWORD command enables any user who has the PASSWORD privilege to alter his own signon password.

FORMAT:

```
(PAS)SWORD CURRENT PASSWORD, NEW PASSWORD
```

PARAMETERS:

CURRENT PASSWORD	must exactly match the user's current account password.
NEW PASSWORD	specifies the new account password. This password replaces the current password in the authorized user file. The password can be up to 12 characters long; remaining characters are truncated. All alphabetic, numeric, and special characters except blanks, commas, or semicolons are allowed.

2.29 INIT COMMAND

The INIT (file initialization) command initializes all data on a contiguous file to 0.

Format:

$$\text{INIT fd}\left[,\begin{Bmatrix}\text{segsize increment}\\ \boxed{1}\end{Bmatrix}\right]$$

Parameters:

fd	is the file descriptor of any unassigned, unprotected, contiguous file.
segsize increment	is the size of the buffer space used. The default is 1kb.

Functional Details:

INIT is implemented with a CSS procedure that loads and starts the File Manager Support Utility as a task.

The INIT command can be entered in command mode.

Examples:

INIT DATA.FIL	Initializes the file DATA.FIL.
INIT DATA2.FIL,50	Initializes the file DATA2.FIL using a 50kb buffer.

2.30 LOAD COMMAND

The LOAD command is used to load a user's task into memory.

Format:

```
LOAD [taskid,] fd[,segsize increment] [,SCTASK]
```

Parameters:

taskid	specifies the name of the task to be loaded.
fd	specifies the file or device the task is being loaded from.
segsize increment	specifies amount of memory in kb (above the memory size) that the task needs for processing. When a task is built (via Link), the OPTION WORK=n command adds additional memory to a task. The size field in the LOAD command overrides the amount of memory specified by Link. The size is accepted in .25kb increments.
SCTASK	specifies that the task is to be loaded as an SPL/32 spooler subcontrol task. See the SPL/32 Administration and Reference Manual for information on subcontrol tasks and their function. If the SPL/32 spooler is not the spooler being used on the system, the attempted use of this parameter will generate an error message.

Functional Details:

In order to maintain CSS compatibility, the taskid (.BG) can be used. However, the system will ignore it. Any valid taskid can be entered but will be ignored.

If a task is loaded from a direct access device, the system first searches the user volume or the specified volume under the user's account. If the file is not found in the search, the system will search the SYS volume in the SYS account if an account or a volume designator was not specified in the LOAD command. Only values that the user does not explicitly specify will subsequently be searched for. If an extension is not specified in the LOAD command, the extension .TSK is assumed. The LOAD command can be entered in command mode.

An error might occur if a user ID under MTM is the same as the ID of a task loaded from the system console. If a load or fd error is displayed, sign off and sign on again with a different user ID.

A privileged user can specify an account number in the fd. All other users can only specify an account class designator (P, G, S).

Examples:

L VOL:CAL	Load the task from file VOL:CAL.TSK.
L PTRP:	Load a task from the paper tape reader punch device.

2.31 LOG COMMAND

The LOG command logs all user input and MTM responses to a specified fd.

Formats:

```
            ┌ ┌ ( NOCOPY ) ┐ ┐   ┌ ( n  ) ┐
LOG [fd]    │,│ {        } │ │ , │ {    } │
            └ └ ( COPY   ) ┘ ┘   └ ( 15 ) ┘

                ┌ ┌ ( NOCOPY ) ┐ ┐   ┌ ( n  ) ┐
SET LOG [fd]    │,│ {        } │ │ , │ {    } │
                └ └ ( COPY   ) ┘ ┘   └ ( 15 ) ┘
```

Parameters:

fd	is the file descriptor of the log file or device. If no fd is specified, logging is terminated. If fd is a file, it must be previously allocated. Files are assigned EWO privileges so that logged output is added to the end of the file. If a log is active when another LOG command is entered, the old log is closed and the new one is initiated.
COPY	specifies that all output is written to both the terminal and the log device.
NOCOPY	specifies that all output (except messages) is written to the log device and not to the terminal. Messages from other users and the operator are written to both the terminal and the log device. If this parameter is omitted, COPY is the default.
n	is a decimal number from 0 through 65,535 specifying the number of lines after which the user log file is to be checkpointed. If this parameter is omitted, the default is 15 lines. If n is specified as 0, no checkpointing will occur.

Functional Details:

The LOG command and the SET LOG command are the same. The command can be entered either way, and both formats perform the same function.

Checkpointing may be done on any type of file. Since indexed files are buffered, checkpointing may be useful at any time the file is being written to. Checkpointing nonbuffered indexed files and extendable contiguous files is useful only if the file is being expanded. Checkpointing to a contiguous file is meaningless (no operation is performed). The LOG command can be entered in command mode, task loaded mode, and task executing mode.

Example:

```
LOG    LOG.FIL,COPY,10
```

2.32 MESSAGE COMMAND

The MESSAGE command sends a message to a specified user.

Format:

$$\text{MESSAGE}\begin{Bmatrix}\text{userid}\\ \text{.OPERATOR}\end{Bmatrix}\text{message}$$

Parameters:

userid	is the name of the user the message is being sent to. This id can be obtained from the DISPLAY USERS command. A userid of .OPERATOR sends a message to the system console.
message	is the text of the message that the user wants to send.

Functional Details:

The user receiving the message receives the userid of the sender as well as the message.

This command can be entered in command mode, task loaded mode, and task executing mode.

Example:

The following message is sent to userid "AVE" from userid "TK". The format of the message sent is:

```
ME AVE HELLO MTM USER
```

The format of the message received is:

```
TK-HELLO MTM USER
```

2.33 MODIFY COMMAND

The MODIFY command modifies the contents of a memory location in the loaded task.

Format:

$$\underline{\text{MO}}\text{DIFY address,}\ \left[\begin{Bmatrix}\text{data}_1\\ \boxed{0}\end{Bmatrix}\right]\ \left[\text{,data}_2\text{,...,data}_n\right]$$

Parameters:

address	is the halfword boundary address at which the contents of memory are to be modified.
data	is a data field consisting of zero to four hexadecimal digits that represent a halfword to be written into memory starting at the location specified by address. Any string of data less than four characters is right-justified and left-zero filled. If the comma is entered but data is omitted, 0 is entered into one halfword.

Functional Details:

This command causes the contents of the halfword location specified by address (modified by any previous BIAS command) to be replaced with data. The modify address must be aligned on a halfword boundary.

The MODIFY command can be entered in task loaded mode and task executing mode.

Any segment (impure, shared, or task common) to which a u-task has write access can be modified. Only the impure segment can be modified for an e-task.

Examples:

BIAS 0 MOD 12F0,4,0,4,0	Modifies four halfwords at location 12F0 to contain 0004 0000 0004 0000.
MOD D0000,4	Modifies the first halfword of the task common linked to the task using segment register D to 4.

2.34 OPTIONS COMMAND

The OPTIONS command allows an MTM user to change the task options of the currently loaded task.

Format:

$$\underline{O}\text{PTIONS} \left[\begin{Bmatrix}\underline{\text{AFP}}\text{AUSE} \\ \underline{\text{AFC}}\text{ONTINUE}\end{Bmatrix}\right] \left[,\begin{Bmatrix}\underline{\text{SVCP}}\text{AUSE} \\ \underline{\text{SVCC}}\text{ONTINUE}\end{Bmatrix}\right] \left[,\underline{\text{NONRES}}\text{IDENT}\right]$$

Parameters:

AFPAUSE	specifies that the task is to pause after any arithmetic fault.
AFCONTINUE	specifies that if the arithmetic fault (AF) trap enable bit is set, a trap is taken. If the bit is not set, the task continues after an arithmetic fault occurs, and a message is sent to the log device.
SVCPAUSE	specifies that SVC 6 is treated as an illegal SVC (applies to background tasks only). If an SVC 6 is executed within a background task, the task is paused.
SVCCONTINUE	specifies that SVC 6 is treated as a NO-OP (applies to background tasks only). If an SVC 6 is executed within a background task, the task is continued.
NONRESIDENT	specifies that the task is to be removed from memory at end of task.

Functional Details:

The OPTIONS command can be entered in task loaded mode.

Example:

```
    OPT   AFC,SVCC
```

2.35 PASSWORD COMMAND

The PASSWORD command enables any MTM users with the PASSWORD privilege (privileged user) to alter their own signon passwords.

Format:

```
    PASSWORD current password, new password
```

Parameters:

current password	must exactly match the user's current account password.
new password	specifies the new account password. This password replaces the current password in the authorized user file. The password can be up to 12 characters long; remaining characters are truncated. All alphabetic, numeric, and special characters except blanks, commas, or semicolons are allowed.

Functional Details:

If a user without the PASSWORD privilege enters the PASSWORD command, a MNEM-ERR message is generated.

2.36 PAUSE COMMAND

The PAUSE command pauses the currently running task.

Format:

```
    PAUSE
```

Functional Details:

Any I/O proceed, ongoing at the time the task is paused, is allowed to go to completion. This command is rejected if the task is dormant or paused at the time it is entered.

The PAUSE command can be entered in task loaded mode and task executing mode.

2.37 PREVENT COMMAND

The PREVENT command suppresses either messages or the task executing prompt (the hyphen (-) is the default) while an interactive task is running.

Format:

$$\text{\underline{PR}EVENT}\left\{\begin{array}{l}\text{\underline{ME}SSAGE}\\ \text{\underline{PR}OMPT}\\ \text{\underline{ET}M}\\ \text{\underline{\$VAR}IABLE}\end{array}\right\}$$

If a user did not input any of these parameters the terminal will receive both messages and task executing prompts. The task executing prompt indicates that either a task or CSS is executing.

Parameters:

MESSAGE	prevents other MTM users from being able to send messages to the user terminal.
PROMPT	suppresses the printing of the task executing prompt (the hyphen (-) is the default) during task executing mode.
ETM	supresses the display of end of task message.
$VARIABLE	disables variable processing on a per user basis.

Functional Details:

If the MTM system includes variable support and the $VARIABLE parameter is entered, the overall performance of MTM increases.

2.38 PRINT COMMAND

The PRINT command sends the file to be printed to the Spooler for subsequent printing.

Format:

```
    PRINT fd [,DEVICE=pseudo device] [,COPIES=n] [,DELETE] [,VFC]
```

Parameters:

fd	is the name of the file to be printed.
DEVICE=	pseudo device specifies the print device. If this parameter is omitted, output is directed to any available print device.
COPIES=	n allows the user to specify the number of copies of the file fd to be output. From 1 to 255 copies can be made. If this argument is omitted, one copy is the default.
DELETE	specifies the file fd is to be deleted after the output operation is completed. If this argument is omitted and the file is not a spool file, the file is retained.
VFC	specifies that vertical forms control is in use. Currently, the card punch driver does not support VFC.

Functional Details:

If the spool option was not selected at OS/32 sysgen time, this command results in an error.

The PRINT command can be entered in command mode, task loaded mode, and task executing mode.

NOTE

If the SPL/32 spooler is in use on the system, the MTM user has additional options available for use with the PRINT command. See the SPL/32 Administration and Reference Manual for a detailed description of these additional options.

2.39 PUNCH COMMAND

The PUNCH command indicates to the Spooler that the specified file is to be punched.

Format:

```
PUNCH fd [,DEVICE=pseudo device] [,COPIES=n] [,DELETE] [,VFC]
```

Parameters:

fd	is the name of the file to be punched.
DEVICE=	pseudo device specifies the name of the pseudo output device. If the DEVICE= parameter is omitted, punch output is directed to any available punch device.
COPIES=	n is the number of copies desired. From 1 to 255 copies can be made. If the COPIES= parameter is omitted, only one copy is output.
DELETE	specifies that the fd is to be deleted after the output operation is performed. If omitted, the file is retained.
VFC	specifies that vertical forms control is in use. Currently, the card punch driver does not support VFC.

Functional Details:

| If the spool option was not selected at OS/32 sysgen time, this command will result in an error.

The PUNCH command can be entered in command mode, task loaded mode, and task executing mode.

NOTE

If the SPL/32 spooler is in use on the system, the MTM user has additional options available for use with the PUNCH command. See the SPL/32 Administration and Reference Manual for a detailed descripion of these additional options.

2.40 RELEASE COMMAND

The $RELEASE command is used to release a new global or new internal variable. It also releases the variable's associated buffer. This command has no effect on local or global variables created with the $SET command.

Format:

$$\underline{\$RELEASE}\ \left\{\begin{array}{l}\underline{G}\text{VARIABLE}\\ \underline{I}\text{VARIABLE}\end{array}\right\}\ \left[,\left\{\begin{array}{c}n_1/n_2\\ n_1\left[,\ldots,n_n\right]\\ \text{ALL}\end{array}\right\}\right]$$

Parameters:

GVARIABLE	indicates that the variables to be released are new global variables.
IVARIABLE	indicates that the variables to be released are new internal variables.
n_1/n_2	specifies that all variables (of the type selected via the preceding parameter) between the range of n_1/n_2 be released. Where n is a decimal number between 1 and the value allowed at MTM sysgen for the selected variable type.
$n_1 \ldots . n_n$	n is a decimal number of a variable (either new global or new internal) or variables to be released. n must be within the range of 1 and the maximum value allowed at MTM sysgen for the selected variable type.
ALL	specifies that all new internal or new global variables be released. This is the default if no specific variable numbers are specified.

Functional Details:

This command may be entered in command mode, task loaded mode, task executing mode, and CSS mode.

In order to reduce buffer overhead, variables that are no longer being used should be released. If this command is directed to a variable that was already released, the command is ignored and no error message is generated.

New internal variables that have a null or zero value are automatically released.

Examples:

$RELEASE GVARIABLE,1/5	All new global variables from 1 through 5 are released.
$RELEASE IVARIABLE,16,19,18,25	The new internal variables numbered 16, 19, 18, and 25 are released.
$RELEASE IVARIABLE,ALL	All new internal variables are released.

NOTE

This command does not release local and global variables created with the $SET command.

2.41 RENAME COMMAND

The RENAME command changes the name of an unassigned, direct access file.

Format:

```
    RENAME oldfd,newfd
```

Parameters:

oldfd	is the current file descriptor of the file to be renamed.
newfd	is the file descriptor of the renamed file.

Functional Details:

The volume id field of the new file descriptor (newfd) may be omitted. A file can only be renamed if its write and read protection keys are 0 (X'0000').

The RENAME command can be entered in command mode, task loaded mode, and task executing mode.

| The user can only rename private files.

Example:

REN VOL:AJM.CUR,AJM.NEW	Renames file AJM.CUR to AJM.NEW on volume VOL.

2.42 REPROTECT COMMAND

The REPROTECT command modifies the protection keys of an unassigned, direct access file.

Format:

```
    REPROTECT fd,new keys
```

Parameters:

fd	is the file descriptor of the file to be reprotected.
new keys	is a hexadecimal halfword whose left byte signifies the new write keys and whose right byte signifies the new read keys.

Functional Details:

Unconditionally protected files can be conditionally reprotected or unprotected.

The REPROTECT command can be entered in command mode, task loaded mode, and task executing mode.

The user can only REPROTECT private files.

2.43 REWIND AND RW COMMANDS

The REWIND and RW commands rewind magnetic tapes, cassettes, and direct access files.

Format:

```
    REWIND [fd,] lu

        or

    RW [fd,] lu
```

Parameters:

fd	is the file descriptor of the device or file to be rewound.
lu	is the logical unit to which the device or file is assigned. If lu is specified without fd, the operation is performed on the lu regardless of what is assigned to it.

Functional Details:

The REWIND and RW commands can be entered in task loaded mode.

Examples:

REW 1	Causes the file or device assigned to lu1 to be rewound.
REW M300:AJM.OBJ,4	Causes file AJM.OBJ, as assigned to lu4 on volume M300, to be rewound.

2.44 RVOLUME COMMAND

The RVOLUME command enables an MTM user to allow/disallow access to a privately owned disk.

Format:

```
                         ADD,    { actno1 [/{RW|RO}] [,...,max actno [/{RW|RO}]] }
                                 { ALL [/{RW|RO}]                                }

RVOLUME voln,         {  REMOVE, { actno1 ,...,max actno }                            }
                                 { ALL                   }

                         USERS [ , { actno           } ]
                                   { actno1 - actno2 }
                                   { 0-max actno     }
```

Parameters:

voln	is the volume name of the restricted disk.
ADD	indicates that the specified accounts will have access to the restricted disk.
actno	is a decimal number from 0 through the maximum account number allowed on the system (limit 65,535) indicating the accounts allowed/disallowed access to the restricted disk. If ALL is specified, accounts 0 through the maximum account number allowed on the system (limit 65,635) have access to the restricted disk.

RW	indicates that the specified account has read/write access to the restricted disk. If this argument is omitted, the default is read only.
RO	indicates that the specified account has read only access to the restricted disk.
REMOVE	indicates that the specified accounts are disallowed access to the restricted disk. If ALL is specified, all accounts having access to the restricted disk are disallowed access with the exception of the owner's account.
USERS	displays all accounts having access to the restricted disk along with the access privileges.

Functional Details:

A disk marked on as a system disk is treated as a restricted disk. Account number 255 is the owner.

The owner of a private disk can allow/disallow other MTM users, the system operator, and other non-MTM tasks access to the restricted disk.

If an owner enters a REMOVE parameter specifying a private account, access will be denied to the disk; the owner can still add accounts, remove accounts, and display accounts that have access, along with the respective access privileges.

For a user with RW access to a restricted disk, accessing private, group, and system files is exactly the same as accessing files on any other disk.

For a user with RO access to a restricted disk, accessing group and system accounts is the same as accessing files on any other disk. Files within the user's private account can only be assigned SRO or ERO. The user cannot allocate, rename, reprotect, or delete any files.

Examples:

•

```
RVOL MTM,U                                                                    |
      00000/RW  00001-00017/RO          00018/RW  00019-00254/RO |
      00255/RW  00256-01023/RO                                                |

RVOL MTM,A,87/RW                                                              |
RVOL MTM,U                                                                    |
      00000/RW  00001-00017/RO          00018/RW  00019-00086/RO |
      00087/RW  00088-00254/RO          00255/RW  00256-01023/RO |

RVOL MTM,U,87                                                                 |
      00087/RW                                                                |

RVOL MTM,R,87                                                                 |
RVOL MTM,U                                                                    |
      00000/RW  00001-00017/RO          00018/RW  00019-00086/RO |
      00088-00254/RO          00255/RW  00256-01023/RO                        |

RVOL MTM,A,87                                                                 |
RVOL MTM,U                                                                    |
      00000/RW  00001-00017/RO          00018/RW  00019-00254/RO |
      00255/RW  00256-01023/RW                                                |

RVOL MTM,U,87-1200      error since account                                   |
ACCT-ERR  POS=87-1200   limit was 1023                                        |
RVOL MTM,U,87-1000                                                            |
00087-00254/RO          00255/RW        00256-01000/RO                        |
```

2.45 SEND COMMAND

The SEND command sends a message to the currently selected task.

Format:

```
SEND message [;]
```

Parameters:

message	is a 1- to 64-character alphanumeric string.

Functional Details:

The message is passed to the selected task the same way as an SVC 6 send message. Following standard SVC 6 procedures, the message consists of an 8-byte taskid identifying MTM as the sender, followed by the user-supplied character string. The message passed to the selected task begins with the first nonblank character following SEND and ends with a carriage return (CR) or semicolon (;) as a line terminator. A message cannot be sent to a task currently rolled out.

The receiving task must have intertask message traps enabled in its TSW and must have an established message buffer area. Refer to the OS/32 Supervisor Call (SVC) Reference Manual for more information on SVC 6.

The SEND command can be entered in task executing mode.

Example:

```
SEND CLOSE LU2,ASSIGN LU3
```

The following is received by the task:

```
.MTM    CLOSE LU2, ASSIGN LU3
```

2.46 SET GROUP COMMAND

The SET GROUP command enables a privileged user to change the group account number associated with the account the user is currently on. This enables a privileged user to specify any account in the system as the current group account. This command is only valid when issued by a privileged user.

Format:

```
    SET GROUP n
```

Parameters:

n	is a decimal number specifying the new group account to be associated with the user's current account. This number must be within the range of 0 and the maximum account number set in AUF (cannot exceed 65,535).

Functional Details:

The SET GROUP command can be entered in command mode, task loaded mode, task executing mode, or from a CSS. If a task is loaded or executing, MTM also modifies the group account number in the task control block (TCB).

If a nonprivileged user enters this command the following message is generated:

```
    MNEM-ERR POS=GROUP
```

A user may not set his group account to 255.

Example:

The user signs on to account 205 (with privilege option). The group account number associated with account 205 is 240. A DISPLAY FILES command of the following format will cause the files in account 240 (account 205's group account) to be displayed:

```
D F -.-/G
```

A privileged user can then change the group account with the SET GROUP command:

```
SET GROUP 220
```

Now the same DISPLAY FILES command will cause the files in account 220 (account 205's new group account) to be displayed.

The new group account association only exists for the length of the current signon session. The group-private account associations specified with the authorized user utility are not changed by this command. The privileged user may change group numbers as many consecutive times as is desired.

2.47 SET KEYOPERATOR COMMAND

The SET KEYOPERATOR command is used to change the operator character used when defining keywords in a CSS call. When entered without parameters, this command will display the current operator character.

Format:

```
    SET KEYOPERATOR [character]
```

Parameters:

character — is any one of the following characters which will be used for defining keywords in CSS calls:

```
=
>
:
%
&
-
#
```

If no character is entered, the current keyword operator is displayed.

Functional Details:

At signon, the default keyword operator is the equal (=) sign. When this operator is changed via the SET KEYOPERATOR command, the new operator remains in effect until signoff or until another SET KEYOPERATOR command is entered.

NOTE

The SET KEYOPERATOR command only changes the operator used when defining keywords in a CSS call. It has no effect on the operator used when referencing keywords within a CSS.

If the character designated as the keyword operator is to be passed as part of a character string in a CSS call, it must be placed within single or double quotes.

If the keyword operator is used in a CSS call and is not within quotes (single or double), and is not a valid keyword assignment, the following error message is generated.

```
    KEYW-ERR POS=

    (x) MUST BE WITHIN 'OR" IF NOT USED AS A KEYWORD OPERATOR.
```

The SET KEYOPERATOR command can be entered in CSS mode, task loaded, task executing, and command mode.

2.48 SET PRIVATE COMMAND

The SET PRIVATE command enables a privileged user to change the private account that the user is currently in without knowing the password of the new account. This enables a privileged user to access any account on the system as their private account. This command is only valid when issued by a privileged user.

Format:

```
SET PRIVATE n
```

Parameter:

n	is a decimal number specifying the new private account number the user wants access to, except account 255. Account 255 can only be accessed via SIGNON. n is within the range of 0 to the maximum account number set in the authorized user file (cannot exceed 65,535).

Functional Details:

The privileges of the user's original signon account remain in effect regardless of the account the user is currently in. A user can neither gain nor lose privileges when moving from account to account.

The SET PRIVATE command can be entered in command mode, task loaded mode, task executing mode, and from a CSS. If a task is loaded or executing when a SET PRIVATE command is entered, MTM also modifies the private account number in the TCB.

If a nonprivileged user enters this command, the following message is generated:

```
MNEM-ERR
```

Example:

The user is signed on to account number 255. A DISPLAY FILES/P command would display all files in account 255. The user changes the current account with a SET PRIVATE command:

```
    SET PRIVATE 210
```

The current account number becomes 210. The group account number remains unchanged. A DISPLAY FILES/P command would display all files in account 210. The user may alter private accounts as often (consecutively) as is desired. Note that account times and usage information used by the accounting reporting utility use the original signon account number regardless of the account the user is in at signoff time.

2.49 SIGNOFF COMMAND

The SIGNOFF command terminates the terminal session. If a user signs off when a task is loaded, the task is cancelled.

Format:

```
    SIGNOFF
```

Functional Details:

When a terminal user signs off the system, these messages are displayed:

```
    ELAPSED TIME=hh:mm:ss          CPUTIME=utime/ostime
    SIGNON LEFT=hh:mm:ss           CPU LEFT=hh:mm:ss
    TIME OFF=mm/dd/yy  hh:mm:ss
```

The SIGNOFF command can be entered in command mode, task loaded mode, and task executing mode. It cannot be followed by another command on the same command line.

2.50 SIGNON COMMAND

The SIGNON command allows a user to communicate with MTM. No commands are accepted until a valid SIGNON command is entered.

Format:

```
                                     ┌                    ┌      ┐ ┐
SIGNON userid,actno,password         │ ,ENVIRONMENT=      │  fd  │ │
                                     │                    │NULL[:]│ │
     [,CPUTIME=maxtime]              └                    └      ┘ ┘
     [,classid=iocount1 [,...,classid=iocount32]]
```

Parameters:

userid	is a 1- to 8-character alphanumeric string specifying the terminal user's identification.
actno	is a 1- to 5-digit decimal number specifying a valid account number (defined in the AUF). If the number is greater than the current account limit (set in the AUF) or is not an established account, an error message is generated.
password	is a 1- to 12-character alphanumeric string specifying the terminal user's password.
ENVIRONMENT=	fd is the file descriptor specifying an existing file that will establish the user's environment at signon time. NULL specifies that the signon CSS routine, USERINIT.CSS, should be ignored and the user will establish the environment after signing on. If the entire keyword parameter is omitted, MTM searches all online disks for the signon CSS procedure USERINIT.CSS/P. The system account, on the system volume, is searched last. If USERINIT.CSS is found, MTM calls the CSS and executes the routine. If it is not found, MTM enters command mode.

If the user does not have the ENV= privilege (privilege to enter ENV= at signon), MTM will ignore this parameter and force the USERINIT.CSS to be executed (if found).

CPUTIME=	maxtime is a decimal number specifying the maximum CPU time to which the session is limited. If this parameter is omitted, the default established at sysgen time is used. If 0 is specified, no limits are applied. The parameter can be specified as: mmmm:ss hhhh:mm:ss ssss
classid=	is one of the 4-character alphanumeric mnemonics specified at sysgen time associated with each specified device or file class.
iocount	is a decimal number specifying the maximum number of I/O transfers associated with a particular device class to which the job is limited. If this parameter is omitted, the default established at sysgen time is used. If 0 is specified, no limits are applied to that class.

Functional Details:

The SIGNON command can be entered in command mode. It cannot be followed by another command on the same line.

When ENVIRONMENT=NULL is specified, the colon is optional. This allows the user the ability to specify the null device (NULL:).

The ENVIRONMENT= parameter may be ignored by the system, depending on the user's account privileges.

Examples:

```
SIGNON ME,12,PASSWD

SIGNON ME,118,SWDOC,ENV=NULL

SIGNON ME,118,SWDOC,ENV=XYZ
```

2.51 SPOOLFILE COMMAND

This command is valid only on systems which are using the SPL/32 spooler. Systems on which the OS/32 spooler is being used may not use the spoolfile command.

The SPOOLFILE command allows a user to allocate a spool file on behalf of a specified pseudo device and assign that file to a specified lu of the currently selected task. This command makes all spooling options available at a terminal or command substitution system (CSS) level.

Format:

```
                                                  ┌ ┌VFC   ┐ ┐
SPOOLFILE lu&lul,pseud dev,FORM=formname          │,┤      ├ │
                                                  └ └IMAGE ┘ ┘

   ┌ ┌NOIMAGE┐ ┐ ┌ ┌CHECKPOINT  ┐ ┐               ┌ ┌HOLD   ┐ ┐
   │,┤       ├ │ │,┤            ├ │  [,COPIES=n]  │,┤       ├ │
   └ └NOVFC  ┘ ┘ └ └NOCHECKPOINT┘ ┘               └ └RELEASE┘ ┘

   ┌,BLOCK= blocksize/indexsize┐ ┌ ┌DELETE  ┐ ┐
   │                           │ │,┤        ├ │  [, PRIORITY=p]
   └             1             ┘ └ └NODELETE┘ ┘
```

Parameters:

lu	is a decimal number specifying the logical unit to which the pseudo device is to be assigned.
lul	indicates that lu is to be assigned to the same spool file as lul. lul must be the first lu assigned to the spool file.
pseud dev	is the 1- to 4-character name of a pseudo device. The first character must be alphabetic; the remaining alphanumeric.
FORM=	is a desired preprinted form name that can be specified here. If the form specified was not previously enabled using a FORM command, an error message is sent to the monitoring control or subcontrol task and the request is processed using the default standard form name, STD.

VFC	specifies the use of vertical forms control for the assigned lu. When VFC is used, the first character of each record is interpreted as a vertical forms control character. If VFC is not included, there is no vertical forms control for the device assigned to the specified lu.
NOVFC	turns the vertical forms control option off for the assigned lu. This is the default option.
COPIES=	identifies the number of copies to be output. It must be between 1 and 255 or an error message is sent.
HOLD	causes the specified file to remain on the spool queue until a RELEASE request is issued.
RELEASE	enables a spool file for output when the lu is closed.
BLOCK	specifies the index and/or data block size.
blocksize	is a decimal number specifying the physical block size in 256-byte sectors, to be used for buffering and debuffering operations involving the file. The default size is 1 or the value entered using the BLOCK command. If this value exceeds the maximum block size established at sysgen time, an error will be printed when attempting to allocate the file.
indexsize	is a decimal number specifying the index block size in 256-byte sectors. The default size is 1 or the value entered using the BLOCK command. Index size cannot exceed the maximum index block size established at sysgen time or an error will occur when attempting to allocate the file.
DELETE	the file is deleted after output. This is the default option.
NODELETE	the file is not deleted after output.
PRIORITY=p	p is the desired print priority. If this option is not specified, the print priority becomes the same as the priority of the task from which the spool file assign originated.

Functional Details:

The SPOOLFILE command can be used to make an assignment to a pseudo device from the terminal or CSS level. If two conflicting parameters are entered in a single SPOOLFILE command, such as DELETE and NODELETE, the second parameter is executed and an error message is generated. The SPOOLFILE command can be entered in task loaded mode.

Example:

```
    SPOOLFILE 4,pdl:,VFC,DELETE
```

This example causes a spool file to be allocated for pseudo device pdl: and assigns that file to logical unit 4 of the current task. Vertical forms control has been specified for the specified lu and the DELETE option has been selected, which means the file will be deleted after output.

2.52 START COMMAND

The START command initiates execution of a dormant task.

Format:

$$\text{START}\left[\begin{Bmatrix}\text{address}\\ \text{transfer address}\end{Bmatrix}\right]\left[,\text{parameter}_1,\ldots,\text{parameter}_n\right]$$

Parameters:

address	specifies the address at which task execution is to begin. For user tasks, this is not a physical address but an address within the task's own program. For executive tasks, it is a physical address. If address is omitted or is 0, the loaded task is started at the transfer address specified when the task was established.
parameter	specifies optional parameters to be passed to the task for its own decoding and processing. All user specified parameters are moved to memory beginning at UTOP. If no parameters are specified, a carriage return is stored at UTOP.

Functional Details:

The START command can be entered in task loaded mode.

Examples:

ST 138	Starts the currently selected task at X'138'.
ST 100,NOSEG,SCRAT	Starts the currently selected task at X'100' and passes NOSEG,SCRAT to the task.
ST ,1000,ABC	Starts the currently selected task at transfer address and passes 1000,ABC to the task.

2.53 TASK COMMAND

The TASK command maintains CSS compatibility of MTM to the operating system. No specific action is performed by this command.

Format:

$$\text{TASK}\left[\begin{Bmatrix}\text{taskid}\\ \underline{\text{.BG}}\text{ROUND}\end{Bmatrix}\right]$$

Parameters:

taskid	is the name of the taskid that has been loaded into the foreground segment of memory.
.BGROUND	indicates that the task has been loaded as a background task.

Examples:

```
T .BG

T COPY
```

2.54 TEMPFILE COMMAND

The TEMPFILE command allocates and assigns a temporary file to an lu for the currently selected task. A temporary file exists only for the duration of the assignment. When a temporary file is closed, it is deleted.

Format:

$$\text{TEMPFILE lu,}\left\{\begin{array}{l} \underline{\text{CO}}\text{NTIGUOUS,fsize} \\ \underline{\text{EC}}\left[/\left[\begin{Bmatrix}\text{bsize}\\ 64\end{Bmatrix}\right]\right] \left[/\left[\begin{Bmatrix}\text{isize}\\ 3\end{Bmatrix}\right]\right] \\ \underline{\text{I}}\text{NDEX}\left[,\left[\begin{Bmatrix}\text{lrecl}\\ 126\end{Bmatrix}\right]\right] \left[/\left[\begin{Bmatrix}\text{bsize}\\ 1\end{Bmatrix}\right]\right] \left[/\left[\begin{Bmatrix}\text{isize}\\ 1\end{Bmatrix}\right]\right] \\ \underline{\text{NB}}\left[,\left[\begin{Bmatrix}\text{lrecl}\\ 126\end{Bmatrix}\right]\right] \left[/\left[\begin{Bmatrix}\text{bsize}\\ 64\end{Bmatrix}\right]\right] \left[/\left[\begin{Bmatrix}\text{isize}\\ 3\end{Bmatrix}\right]\right] \end{array}\right\}$$

Parameters:

lu	is a decimal number specifying the lu number to which a temporary file is to be assigned.
CONTIGUOUS	specifies that the file type to be allocated is contiguous.
fsize	is a decimal number specifying the total allocation size in 256-byte sectors. This size can be any value up to the number of contiguous free sectors existing on the specified volume at the time the command is entered.
EC	specifies that the file type to be allocated is extendable contiguous.

bsize	is a decimal number specifying the physical block size to be used for buffering and debuffering operations. bsize represents the block size in sectors of the physical data blocks containing the file. For INDEX files, this parameter cannot exceed the maximum block size established by the system generation (sysgen) procedure. For EC and NB files, this parameter may be any value between 1 and 255 inclusive. If bsize is omitted, the default value for INDEX files is 256 bytes (one sector). For EC and NB files, the default is 64 sectors.
isize	is a decimal number specifying the index block size. For INDEX files, the default value is one sector (256 bytes). For EC and NB files, the default value is three sectors (768 bytes). The index block size cannot exceed the maximum disk block size established by the sysgen procedure. isize may not exceed 255.
INDEX	specifies that the file type to be allocated is indexed.
lrecl	is a decimal number specifying logical record length in bytes. It cannot exceed 65,535 bytes; its default is 126. The logical record length is meaningful only for indexed and nonbuffered indexed files.
NB	specifies that the file type to be allocated is nonbuffered indexed.

Functional Details:

A temporary file is allocated on the temporary volume.

To assign this file, sufficient room must exist in system space for three buffers, each of the stated size. Therefore, if bsize or isize is very large, the file cannot be assigned in some situations. A maximum block size parameter is established in the system at sysgen time. The bsize and isize cannot exceed this constant.

To assign an EC or NB file, sufficient room must exist in system space to contain only the index block of the stated size. The data blocks for EC and NB files are not buffered in system space and thus are not constrained to the sysgened block size.

The TEMPFILE command can be entered in task loaded mode and task executing mode.

Examples:

TE 2,CO,64	Allocates, on the temporary volume, a contiguous file with a total length of 64 sectors (16kb) and assigns it to the loaded task's lu2.
TE 14,IN,126	Allocates, on the temporary volume, an index file with a logical record length of 126 bytes. The buffer size and index block size default to one sector. The file is assigned to lu14 of the loaded task.
TE 5,EC	Allocates, on the temporary volume, an extendable contiguous file with default data block size of 64 and index block size of 3 sectors. The file initially contains no records, and has a record length of one sector (same as a contiguous file). The file is assigned to lu5 of the task.
TE 7,NB,240/250/5	Allocates, on the default temporary volume, a temporary nonbuffered indexed file with logical record length of 240 bytes, data block size of 250 sectors, and index block size of 5 sectors. The file initially contains no records. The file is assigned to lu7 of the task.

2.55 VOLUME COMMAND

The VOLUME command sets or changes the name of the default user volume. It may also be used to query the system for the current names associated with the user, system, roll, spool, or temporary volume.

Format:

```
    VOLUME  voln
```

Parameter:

voln	is a 4-character volume identifier. If this parameter is omitted, all current default user, system, roll, spool, and temporary volume names are displayed.

Functional Details:

Any commands that do not explicitly specify a volume name use the default user volume. No test is made to ensure that the volume is actually online at the time the command is entered. If voln is not specified, the names of the current default volumes are output to the user console.

The default user volume is initially set to the system volume or the default user volume (set at MTM sysgen time) when the user signs on. If no volume was specified at MTM sysgen, the default is the system volume. This command may be entered in command mode, task executing mode, and task loaded mode.

Example:

```
    VOL
    USR=MTM   SYS=MTM   SPL=M67B   TEM=M301   RVL=MTM
```

When MTM is used in conjunction with the new spooler, SPL/32, the spool volume is not displayed by the VOLUME command.

Example:

```
VOL
USR=MTM    SYS=MTM    TEM=M301    RVL=MTM
```

2.56 WFILE COMMAND

The WFILE command writes a filemark on magnetic tapes, cassettes, and direct access files.

Format:

```
WFILE [fd,] lu
```

Parameters:

fd	is the file descriptor of the file or device to which a filemark is to be written.
lu	is the lu to which the device or file is assigned. If lu is specified without fd, the operation is performed on the specified lu regardless of what is assigned to it.

Functional Details:

The WFILE command can be entered in task loaded mode.

Examples:

WF 1	Causes a filemark to be written on the device or file assigned to lu1.
WF M300:AJM.OBJ,4	Causes a filemark to be written on file AJM.OBJ, which is assigned to lu4 on volume M300.

2.57 XALLOCATE COMMAND

The XALLOCATE command is used to create a direct access file.

Format:

```
                 { CONTIGUOUS,fsize [,{keys}]
                 {                    {0000}
                 { EC [/[{bsize}]] [/[{isize}]] [,[{keys}]]
                 {        { 64  }      {  3  }      {0000}
XALLOCATE fd,   <  INDEX [,[{lrecl}]] [/[{bsize}]] [/[{isize}]] [,[{keys}]]
                 {           { 126 }      {  1  }      {  1  }      {0000}
                 { NB [,[{lrecl}]] [/[{bsize}]] [/[{isize}]] [,[{keys}]]
                 {        { 126 }      { 64  }      {  3  }      {0000}
                 { ITAM [,[{lrecl}]] [/[{bsize}]] [[,{keys}]]
                 {          { 80  }      {  1  }      {0000}
```

Parameters:

fd	is the file descriptor of the file to be allocated.
CONTIGUOUS	specifies that the file type to be allocated is contiguous.
fsize	is a decimal number indicating file size which is required for contiguous files. It specifies the total allocation size in 256-byte sectors. This size may be any value up to the number of contiguous free sectors existing on the specified volume at the time the command is entered.

keys	specifies the write and read protection keys for the file. These keys are in the form of a hexadecimal halfword, the left byte of which signifies the write key and the right byte the read key. If this parameter is omitted, both keys default to 0.
EC	specifies that the file type to be allocated is extendable contiguous.
bsize	is a decimal number specifying the physical block size to be used for buffering and debuffering operations on indexed files or data communications devices. When INDEX, EC, or NB is specified, bsize represents the block size in sectors of the physical data blocks containing the file. When ITAM is specified, bsize represents the buffer size in bytes. For INDEX files and ITAM buffers, this parameter cannot exceed the maximum block size established by the system generation (sysgen) procedure. For EC and NB files, this parameter may be any value between 1 and 255 inclusive. If bsize is omitted, the default value for INDEX files and ITAM buffers is 256 bytes (one sector). For EC and NB files, the default is 64 sectors.
isize	is a decimal number specifying the index block size. For INDEX files, the default value is one sector (256 bytes). For EC and NB files, the default value is three sectors (768 bytes). The index block size cannot exceed the maximum diskblock size established by the sysgen procedure. Neither bsize nor isize may exceed 255.
INDEX	specifies that the file type to be allocated is indexed.
lrecl	is a decimal number specifying the logical record length of an indexed file or communications device. It cannot exceed 65,535 bytes. Its default is 126 bytes. It may optionally be followed by a slash (/) which delimits lrecl from bsize.
NB	specifies that the file type to be allocated is extendable contiguous.
ITAM	specifies that the fd to be allocated is an ITAM buffered communications device.

Functional Details:

The XALLOCATE command is different from the ALLOCATE command in that if fd is an existing file, it is deleted and reallocated. If fd does not exist, it is allocated.

If the fd to be allocated is a device name instead of a filename, a DEL-ERR TYPE=VOL occurs.

The XALLOCATE command can be entered in command mode, task loaded mode, and task executing mode.

2.58 XDELETE COMMAND

The XDELETE command is used to delete one or more files. If the file does not exist, no error is generated.

Format:

$\underline{\text{XD}}\text{ELETE } fd_1 \left[, fd_2 \ldots, fd_n\right]$

Parameter:

fd	is the file descriptor of the file to be deleted.

Functional Details:

A file can only be deleted if it is not currently assigned to a task and its write and read protection keys are 0 (X'0000').

| An MTM user can only delete private files.

Example:

```
XDEL FIXD:OS323240.817,RADPROC.FTN
```

CHAPTER 3
MTM/NON-MTM TASK INTERFACES

3.1 INTRODUCTION

Multi-Terminal Monitor (MTM) allows the terminal user to transfer control of a terminal to tasks other than MTM and then return the terminal to MTM control in an orderly fashion. This orderly transfer of control is accomplished via the use of interface protocols that are invoked by specific MTM commands. The MTM terminal user can interface with:

- foreground tasks,
- HASP tasks, and
- ITC/RELIANCE tasks.

3.2 INTERFACING WITH A FOREGROUND TASK

The foreground interface allows an MTM user to connect an MTM terminal to any specified foreground task selected via the following command:

Format:

```
    $foreground task-id
```

Parameter:

foreground task-id	is a task-id of 1- to 7-characters specifying the selected foreground task to which the MTM terminal is to be connected. The following task-ids are restricted and cannot be used: HASP .MTM .SPL ECM

Functional Details:

This feature is available to all MTM users that have the $foreground privilege.

This command can be entered in command mode as long as no CSS is active. This command is not available in batch mode. While a terminal is connected to a foreground task, all MTM messages to that terminal are ignored.

The foreground task to which this command is directed must have particular characteristics and options enabled in order to establish, maintain, and terminate the interface. The foreground task must be linked with option UNIVERSAL and must be able to send and receive messages via SVC 6. For further information regarding SVC 6 use, refer to the OS/32 Supervisor Call (SVC) Reference Manual.

Example:

```
    $XYZ
```

In this example, the MTM terminal issuing the $XYZ command becomes connected to the foreground task identified as XYZ.

A subsequent DISPLAY USERS command from an MTM terminal will display the terminal transferred to the foreground task's (XYZ) control as shown:

```
    DAVE - NULL:@$XYZ
```

3.2.1 Programming Details

The foreground task selected with the $FGRND command must have the following interface and a message buffer ring with message entries enabled. The task-id may have no more than seven characters.

The selected task gets the following message from .MTM:

```
    ADD terminal-dn,priv-acc,group-acc,userid <CR>
```

The foreground task must now assign the terminal with terminal-dn and immediately send the following message to .MTM:

```
    $STA terminal-dn, status <CR>
```

To return the terminal to MTM control, the foreground task should close the terminal and send the following message to .MTM:

```
$END terminal-dn<CR>
```

MTM assigns the terminal and the user returns to MTM control.

Parameters:

terminal-dn	device name of the user's terminal (variable length from two to five characters including ":")
priv-acc	user's private account number (fixed length of five characters, right justified, leading zeros.)
group-acc	user's group account number (fixed length of five characters, right justified, leading zeros)
userid	userid under MTM (fixed length of eight characters left justified)
status	returned from foreground task:

X'30'	all OK - foreground task has assigned the terminal.
X'31'	assign-errors - terminal was not assigned by the foreground task (.MTM reports TASE-ERR to the user).
X'39'	space error - terminal would have exceeded the maximum number of allowed terminals. (.MTM reports TSPC-ERR to the user.)

<CR>	carriage return (X'OD')

Functional Details:

Every ten seconds, MTM tries to reassign the terminal; i.e., if the foreground task closes the terminal or goes to end of task without sending a $END message, the user terminal remains unassigned no longer than 10 seconds.

3.3 HASP INTERFACE

The HASP interface allows an MTM user to communicate with a specified HASP control task in the foreground. The option for the HASP interface must be enabled at MTM sysgen in order for it to be available to MTM users. When the HASP task is started, the optional start parameter OUT=/MTM must be used to allow messages to be output to MTM.

Format:

```
$HASPxx
```

Parameters:

xx	is a two-character alphanumeric extension of the HASP control tasks foreground id.

Functional Details:

Option UNIVERSAL is required when linking the HASP task. Once the $HASP command has been executed, the MTM terminal is then in HASP mode. The HASP mode read prompt is:

```
"
```

All commands entered on the terminal are sent to the specified HASP task. All commands starting with a $ are prefixed with the HASP message command and then sent to the specified HASP task. All messages sent by HASP to the terminal are displayed in the following format:

```
HASPxx> message.....
```

When the user is ready to return the terminal to MTM control, the following command is used:

```
$MTM
```

The terminal is then returned to MTM control.

The $HASPxx command can be entered in command mode only. No task can be loaded or executing, no CSS active, and the user must not be in batch mode. While in HASP mode MTM messages from other users and the system operator can be displayed on the HASP terminal.

The specified HASP task is set to the same private and group account number as the user. If $MTM is entered, the specified HASP task remains on these accounts and continues sending messages to the user terminal until another user connects to the same HASP task or until signoff.

Example:

```
$HASP03
```

This example selects the HASP task with the taskid HASP03 in the foreground. The terminal is now in HASP mode (if no errors occurred.)

3.4 ITC/RELIANCE INTERFACE

The environmental control monitor (ECM) provides facilities for terminal users to transfer control of their terminals between Reliance and MTM, or between different Reliance environments, without use of the system console or a Reliance controller's terminal. For details about the use of the ECM, refer to the Environmental Control Monitor/32 (ECM/32) Systems Programming and Operations Manual.

CHAPTER 4
PROGRAM DEVELOPMENT

4.1 INTRODUCTION

This chapter is written as a program development tutorial session for new to intermediate users. The program development commands enable you to easily create a program and modify, maintain, and execute it from the terminal.

4.2 CREATING A SOURCE PROGRAM

To create a source program that will exist in a single source file (language environment), enter a program development language command with a user-specified filename. Source filename extensions are program-supplied and language dependent. The language command entered must be consistent with the language of the source file. When a language command is entered, a file is allocated (if it does not already exist) with the user-specified filename and program-supplied filename extension, and the editor is loaded and started. If the file exists, it is set as the current program (EDIT is not loaded.)

Table 4-1 lists the program development language command syntax and program-supplied filename extensions.

TABLE 4-1 PROGRAM DEVELOPMENT LANGUAGE COMMANDS

LANGUAGE	COMMAND SYNTAX	PROGRAM DEVELOPMENT FILENAME EXTENSIONS
CAL/32	CAL [[voln:] filename]	.CAL
CAL Macro/32	MACRO [[voln:] filename]	.MAC
FORTRAN VII	FORT [[voln:] filename] (using development compiler)	.FTN
FORTRAN VII	FORTO [[voln:] filename] (using optimizing compiler)	.FTN

TABLE 4-1 PROGRAM DEVELOPMENT LANGUAGE COMMANDS (Continued)

LANGUAGE	COMMAND SYNTAX	PROGRAM DEVELOPMENT FILENAME EXTENSIONS
FORTRAN VII	FORTZ [[voln:] filename] (using the universal compiler)	.FTN
COBOL	COBOL [[voln:] filename]	.CBL
REPORT PROGRAM GENERATOR	RPG [[voln:] filename]	.RPG
Pascal	PASCAL [[voln:] filename]	.PAS

Program development language commands automatically set up certain processes that will be used for the remainder of the development effort. These processes are:

- Assignment of the standard source file language extensions,
- The compiler or assembler to be used,
- The standard Perkin-Elmer run time libraries to be linked, and
- The language tab character, a back slash, (\), and tab settings pertinent to the specified language, (displayed when the editor is entered).

These automatic specifications free you from constantly having to remember them. The user-supplied filename with the program-supplied extension will identify the source file throughout the program development session.

Once the editor is loaded and started, the full range of Edit commands are available to create the source file. See the OS/32 Edit User Guide.

Example:

```
*FORT PROG1
** NEW PROGRAM
-EDIT
-G PROG1.FTN
-O TA = \,7,73
-O COM = CON:
    .
    .
    .
(edit session)
    .
    .
    .
>SAVE*
>END
-WORK FILE = M67B: PROG1.000/P
-RENUMBERED INPUT FILE AVAILABLE, M67B: PROG1.FTN/P
```

In this example, the FORTRAN language command entered with a user-supplied filename allocates an empty file, PROG1.FTN, and loads and starts the editor. The FORTRAN tab settings are set and displayed. The filename you specify is set as the current program and is always accessed and/or executed if you do not specify another filename. You can start to enter your program after these messages are displayed:

```
** NEW PROGRAM
-EDIT
>
```

You can also create a source file by entering a language command without a filename. Then enter the EDIT command with a filename. The EDIT command allocates a file and loads and starts the editor. You can employ all of the Edit commands to create your source file.

Example:

```
*FORT
*EDIT PROG1
-EDIT - PROG1.FTN
    .
    .
    .
(edit session)
    .
    .
    .
>SAVE*
>END
```

The FORT command creates the language environment. The EDIT command entered with PROG1 loads and starts the editor and allocates PROG1.FTN for the source file that will be created via the Edit commands. PROG1.FTN is saved and the edit session is ended.

4.2.1 Creating a Data File

To create a data file, save the source program file to disk, and clear the edit buffer by deleting all lines currently in the buffer.

Example:

```
>SAVE*
>DELETE 1-
>AP
    .
    .
    .
(use the editor to create PROG1.DTA)
    .
    .
    .
>SAVE PROG1.DTA
>END
```

In this example, PROG1.FTN is saved and then cleared from the edit buffer. The Edit APPEND command allows data to be entered in the data file. The data file is saved, and the edit session is terminated with the END command.

4.3 EXECUTING A PROGRAM

The program development EXEC command loads and runs the current program.

Example:

```
*EXEC
** EXECUTION OF PROG1.FTN FOLLOWS:
-END OF TASK CODE=0
```

This example assumes that PROG1.FTN already exists as the current program. The EXEC command loads and runs the current program, PROG1.FTN, and displays a zero end of task code (if no errors occurred). A nonzero end of task code indicates an error was encountered.

4.4 MODIFYING A PROGRAM

To modify your program, enter the appropriate language command with the filename of the source file to be modified. Enter the EDIT command to access the editor.

Example:

```
    *FORT PROG1
    *EDIT
    -EDIT - PROG1.FTN
        .
        .
        .
    (edit session to modify PROG1)
        .
        .
        .
    >SAVE*
    >END
```

In this example, the FORTRAN language command is entered with the filename PROG1. The editor is accessed via the EDIT command, and the name of the current program is displayed. The editor is used to modify the source file, PROG1.

4.5 RE-EXECUTING A MODIFIED PROGRAM

When the EXEC command is issued, the source program is compiled, linked, and executed, creating object and image modules. If the source file is subsequently modified, the dates assigned to the previously compiled object and previously linked image modules will not be current.

Dates and times are assigned to source, object, and image modules when they are created. The dates are stored in the system directory.

The EXEC command causes the object and image modules to be datechecked. They are then compiled and/or linked if they are out of date. The EXEC command then loads and runs the image program.

Example:

```
    *EXEC PROG1
    -FORTRAN PROG1.FTN
    -END OF TASK CODE=0
    -LINK PROG1.OBJ
    -END OF TASK CODE=0
    ** EXECUTION OF PROG1 FOLLOWS:
    -END OF TASK CODE=0
```

This example assumes that PROG1.FTN already exists. The EXEC command, entered with PROG1, compiles, links, and then executes the image program. A zero end of task code is displayed after each process if no errors were encountered.

The program development RUN command can also be used to execute a program. The RUN command does not datecheck, compile, or link. It simply runs a program that was already compiled and linked.

Example:

```
*RUN PROG1
** EXECUTION OF PROG1 FOLLOWS:
-END OF TASK CODE=0
```

If the EXEC or the RUN command is entered without a filename, the current program is executed. If there is no current program, this message is displayed:

```
** CURRENT PROGRAM NOT SPECIFIED
```

If you only want to compile a program without linking or executing it, the program development COMPILE command can be used. The program development COMPLINK command compiles and links a program, if necessary, but does not execute it. The program development LINK command links the object program but does not execute it. These commands are explained fully in their respective sections.

4.6 EXECUTING MULTIPLE PROGRAMS AS A SINGLE PROGRAM

If a source program exists in multiple source files (multi-module environment), you must include the file descriptors (fd) of each source file in an environment descriptor file (EDF). The EDF retains the identity of all the source files in the multi-module environment that will be used to create a program.

When you enter the program development ENV command, you indicate that your source program exists in more than one file and is to be created in a multi-module environment. The ENV command creates the multi-module environment and allocates an EDF to contain the fds of the source files.

Example:

```
*ENV ALLPROG
** NEW ENVIRONMENT
```

In this example the ENV command with the user-specified EDF name, ALLPROG, creates the multi-module environment.

No language extension is specified with the EDF filename since each module can be written in a different language. Attempting to enter an extension will cause an error. The user-specified or default volume is searched for ALLPROG. If it is not found, an empty file named ALLPROG is allocated, and the message, NEW ENVIRONMENT, is displayed. The EDF is now ready to receive the fds of the multiple source files. The program development ADD command is used to add source program fds to the the multi-module environment.

Example:

```
*ENV ALLPROG
** NEW ENVIRONMENT
*ADD PROG1.FTN
*ADD PROG2.CBL
```

The multi-module environment is created and an EDF, ALLPROG, is allocated via the ENV command. The ADD command adds the fds, PROG1.FTN and PROG2.CBL, to the multi-module environment.

When the ADD command is entered with a user-specified fd, the EDF is searched for that fd. If the fd does not already exist in the multi-module environment, it is added. If it already is in the multi-module environment, this message is displayed:

```
** FILENAME CONFLICT - ENTRY NOT ADDED
```

You must rename the file or remove the existing entry from the environment.

The program development LIST command displays the fds in the multi-module environment, and the program development REMOVE command removes fds from the multi-module environment.

Example:

```
*LIST
** CURRENT ENVIRONMENT = ALLPROG
-PROG1.FTN
-PROG2.CBL
*REMOVE PROG2
*LIST
** CURRENT ENVIRONMENT = ALLPROG
-PROG1.FTN
*EXEC
** EXECUTION OF ALLPROG FOLLOWS:
** END OF TASK CODE=0
>
```

The LIST command displays PROG1.FTN and PROG2.CBL as the fds in the multi-module environment. The REMOVE command removes PROG2. CBL and the LIST command displays the contents of the multi-module environment. The EXEC command runs the program, ALLPROG.

If the ADD or REMOVE command is entered without an fd or if the fd is incorrect, this message is displayed:

```
** SYNTAX ERROR
```

Not all program development commands are available in both language and multi-module environments. Table 4-2 shows the commands that are available in the environments.

TABLE 4-2 PROGRAM DEVELOPMENT COMMAND AVAILABILITY

COMMAND	LANGUAGE	MULTI-MODULE
ADD		x
COMPILE	x	x
COMPLINK	x	x
EDIT	x	x
ENV		x
EXEC	x	x
LINK	x	x
LIST		x
REMOVE		x
RUN	x	x

If a command that is meaningful only in a multi-module environment is entered in a language environment, this message is displayed:

```
** NOT IN MULTI-MODULE ENVIRONMENT
```

In order to re-access a source program, modify the source file, and include it in a multi-module environment, enter the ENV command followed by the EDIT command and use the editor to modify the source file.

Example:

```
*ENV ALLPROG
*ADD PROG1.FTN
*LIST
** CURRENT ENVIRONMENT = ALLPROG
-PROG2.CBL
-PROG1.FTN
*EDIT PROG1.FTN
-EDIT PROG1.FTN
    .
    .
    .
(edit session)
    .
    .
    .
>SAVE*
>END
*EXEC
-PERKIN-ELMER OS/32 LINKAGE EDITOR 03/242 R00-01
-END OF TASK CODE = 0
** EXECUTION OF ALLPROG FOLLOWS:
-END OF TASK CODE = 0
>
```

The multi-module environment is entered via the ENV command and the EDF name, ALLPROG. PROG1.FTN is added to the multi-module environment. The LIST command displays the filenames remembered in the EDF. The EDIT command accesses the editor to modify PROG1.FTN. When the edit session is ended, the EXEC command executes all the modules as one program, displaying an end task code of 0 after successful execution (if no errors were encountered).

4.7 HOW TO RECOVER FROM ERRORS

If an error occurs in program compilation or execution, the process aborts, and a nonzero end of task code and an error message are displayed.

Example:

```
** COMPILE ERRORS, LISTING ON PR:
```

Program development makes it easy for the user to recover from errors. Compile errors are printed in the listing of the source file containing the error.

Use the editor to correct the error and re-execute the program. The EXEC command will recompile only the modified modules.

In some instances the EXEC command will recompile a successfully compiled module if the time between the creation of the source and object is less than one minute.

See the OS/32 Link Reference Manual for an explanation of link error messages.

4.8 ASSIGNING LOGICAL UNITS

Program development defines and sets global variables that are associated with particular devices. These devices have default logical unit (lu) assignments. The global variable names and settings are displayed when the user signs on. Table 4-3 shows the variable names, their default settings, and lu assignments.

TABLE 4-3 PROGRAM DEVELOPMENT DEFAULT VARIABLE SETTINGS AND LU ASSIGNMENTS

VARIABLE NAME	DEVICE	LOGICAL UNIT
SSYSIN	CON:	1
SSYSOUT	CON:	2
SSYSPRT	PR:	3
SSYSCOM	CON:	5
SSYSMSG	CON:	7

Before running a program, ensure that the default variable and lu settings are appropriate. The input device can be changed from the console (default) to a pre-allocated file.

Example:

```
   *SSYSIN FILE.IN
```

Listings can be sent directly to a file rather than to the printer (default).

Example:

```
   *SSYSPRT FILE.OUT
```

The user has the option to specify lu assignments unique to a particular session. This is accomplished by creating a file, via the editor, that contains the new lu assignments. This file must be saved with the extension .ASN, and the last line in the file must be a $EXIT statement. The program development software will first search for a file with the extension .ASN. If no file is found, the default lu assignments are used. The HELP command provides all the information needed to create a new assignment file.

Any variable settings you change supercede the default variable settings and are in effect until you change them again or sign off.

4.9 PROGRAM DEVELOPMENT COMMANDS

This section describes the functions of each of the following program development commands:

- ADD
- COMPILE
- COMPLINK
- EDIT
- ENV
- EXEC
- LINK
- LIST
- REMOVE
- RUN

4.9.1 ADD Command

The ADD command adds the fds of source programs to the multi-module environment. These fds are remembered in the EDF. The ADD command is valid in the multi-module environment only.

Format:

```
    ADD fd [,cssprod]
```

Parameters:

fd	is the file descriptor of the source file to be added to the multi-module environment.
cssprod	is the name of the CSS procedure to be used when nonstandard compilation is required.

Functional Details:

The ADD command causes the current EDF to be searched for the specified fd. If the specified fd is not found, it is added to the multi-module environment. If the fd currently exists in the environment, the following message is displayed:

```
    ** FILENAME CONFLICT - ENTRY NOT ADDED
```

If the fd is omitted, or is in an incorrect format, this message is displayed:

```
    ** SYNTAX ERROR
```

If the fd is entered without an extension, this message is displayed:

```
    ** EXTENSION OMITTED
```

The cssprod parameter must be used if the extension of the specified file differs from the language extensions listed in Table 4-1. If this parameter is omitted when you are using a nonstandard extension, the following messages are displayed:

```
** NONSTANDARD EXTENSION
** ALTERNATE CSS REQUIRED
```

The alternate CSS cannot be specified by just a volume name. It must contain at least a filename.

4.9.2 COMPILE Command

The COMPILE command compiles a source module and creates an object module if an up-to-date object module does not already exist in the language environment. The COMPILE command conditionally compiles when the ALL parameter is specified in the multi-module environment. The COMPILE command does not execute a program.

Language Format:

$$\text{COMPILE}\left[\begin{Bmatrix}\text{voln:}\\ \underline{\text{user voln:}}\end{Bmatrix}\right]\left[\begin{Bmatrix}\text{filename}\\ \underline{\text{current program}}\end{Bmatrix}\right]$$

Multi-Module Format:

$$\text{COMPILE}\left[\begin{Bmatrix}\text{voln:}\\ \underline{\text{user voln:}}\end{Bmatrix}\right]\left[\begin{Bmatrix}\text{filename}\\ \text{ALL}\\ \underline{\text{current program}}\end{Bmatrix}\right]$$

Parameters:

voln:	is a 1- to 4-character alphanumeric name specifying the volume on which the source file resides. If this parameter is omitted, the default is the user volume.
filename	is a 1- to 8-character alphanumeric name specifying the source file. If this parameter is omitted, the current program is the default.
ALL	specifies that all files in the multi-module environment whose fds are remembered in the EDF are to be compiled, if necessary. When this parameter is specified, the COMPILE command conditionally compiles all the files that are in the multi-module environment.

Functional Details:

A successful compilation ends with a zero end of task code. An end of task code other than zero indicates a compilation error that will be printed on the listing created as a result of compile.

If the environment is not set when you enter the COMPILE command, this message is displayed:

```
** ENVIRONMENT NOT SET
```

If a filename is not entered and a current program is not specified, this message is displayed:

```
** CURRENT PROGRAM NOT SET
```

If a specified filename does not exist, the following message is displayed:

```
** file NOT FOUND
```

The COMPILE command functions are illustrated in Figures 4-1 and 4-2.

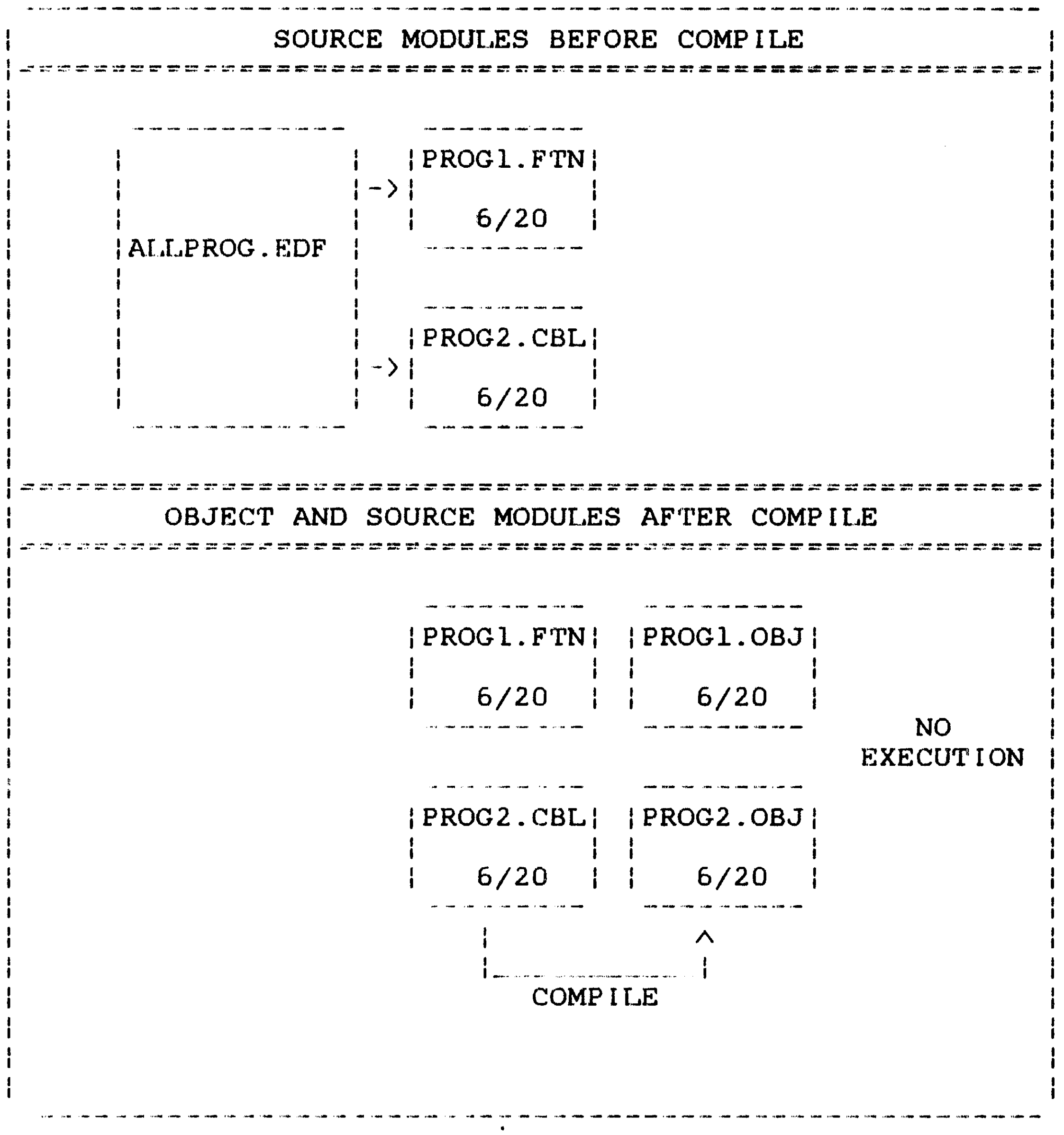


Figure 4-1 COMPILE Command Functions in the Language Environment

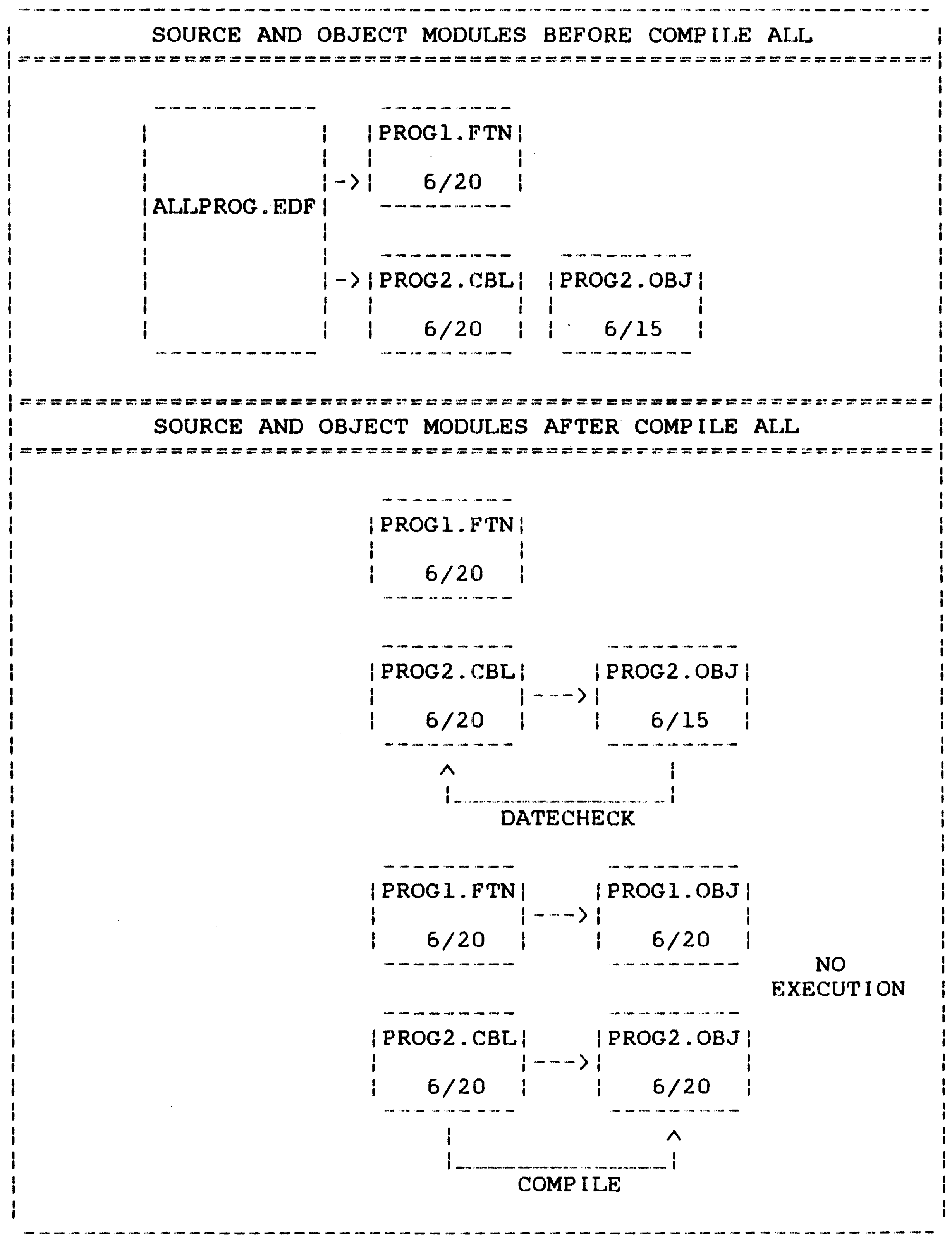


Figure 4-2 COMPILE Command Functions in the Multi-Module Environment

4.9.3 COMPLINK Command

The COMPLINK command performs a conditional compile and a conditional link by datechecking source, object, and image modules in language and multi-module environments. If all modules are up-to-date, this command does not perform any function. This command does not execute the program.

Language Format:

$$\text{COMPLINK}\left[\begin{Bmatrix}\text{voln:}\\ \text{user voln:}\end{Bmatrix}\right]\left[\begin{Bmatrix}\text{filename}\\ \text{current program}\end{Bmatrix}\right]$$

Multi-Module Format:

```
COMPLINK
```

Parameters:

voln:	is a 1- to 4-character alphanumeric name specifying the volume on which the source file resides. If this parameter is omitted, the default is the user volume.
filename	is a 1- to 8-character alphanumeric name specifying the source file. If this parameter is omitted, the current program is the default. Filename specification is meaningful in a language environment only.

Functional Details:

When the COMPLINK command is used in a multi-module environment, all the fds contained in the EDF are datechecked, compiled if necessary, and linked.

If the specified source file is not found, the COMPLINK sequence terminates, and this message is displayed:

```
** fd NOT FOUND
```

If you specify any arguments in a multi-module environment, this message is displayed:

```
** TOO MANY ARGUMENTS
```

The COMPLINK command functions are shown in Figures 4-3 and 4-4.

```
 -----------------------------------------------------------
|   SOURCE, OBJECT, AND IMAGE MODULES BEFORE COMPLINK      |
|==========================================================|
|                                                          |
|        ----------   ----------   ----------              |
|       |PROG4.CBL|  |PROG4.OBJ|  |PROG4.TSK|              |
|       |         |  |         |  |         |              |
|       |  6/20   |  |  6/15   |  |  6/15   |              |
|        ----------   ----------   ----------              |
|                                                          |
|                                                          |
|==========================================================|
|    SOURCE, OBJECT, AND IMAGE MODULES AFTER COMPLINK      |
|==========================================================|
|                                                          |
|                                                          |
|        ----------   ----------   ----------              |
|       |PROG4.CBL|  |PROG4.OBJ|  |PROG4.TSK|              |
|       |         |  |         |  |         |              |
|       |  6/20   |  |  6/15   |  |  6/15   |              |
|        ----------   ----------   ----------              |
|           ^             ^             |                  |
|           |_____________|_____________|                  |
|                   DATECHECK                              |
|                                                          |
|        ----------   ----------   ----------              |
|       |PROG4.CBL|  |PROG4.OBJ|  |PROG4.TSK|     NO       |
|       |         |->|         |->|         |  EXECUTION   |
|       |  6/20   |  |  6/20   |  |  6/20   |              |
|        ----------   ----------   ----------              |
|           |             ^             ^                  |
|           |_____________|_____________|                  |
|              COMPILE         LINK                        |
|                                                          |
|                                                          |
 -----------------------------------------------------------
```

Figure 4-3 COMPLINK Command Functions in the Language Environment

```
 ------------------------------------------------------------------
|        SOURCE, OBJECT, AND IMAGE MODULES BEFORE COMPLINK         |
|==================================================================|
|                                                                  |
|   -----------    ---------    ---------                          |
|  |           |  |PROG1.FTN|  |PROG1.OBJ|                         |
|  |           |  |         |  |         |                         |
|  |ALLPROG.EDF|->|  6/20   |  |  6/10   |                         |
|  |           |   ---------    ---------                          |
|  |           |                                                   |
|  |           |   ---------    ---------                          |
|  |           |->|PROG2.CBL|  |PROG2.OBJ|                         |
|  |    6/8    |  |         |  |         |                         |
|  |           |  |   6/20  |  |   6/10  |                         |
|   -----------    ---------    ---------                          |
|                                                                  |
|==================================================================|
|         SOURCE, OBJECT, AND IMAGE MODULES AFTER COMPLINK         |
|==================================================================|
|                                                                  |
|                  ---------    ---------    -----------           |
|                 |PROG1.FTN|  |PROG1.OBJ|  |           |          |
|                 |         |  |         |->|           |          |
|                 |  6/20   |  |  6/10   |  |ALLPROG.TSK|          |
|                  ---------    ---------   |           |          |
|                  ---------    ---------   |           |          |
|                 |PROG2.CBL|  |PROG2.OBJ|->|           |          |
|                 |         |  |         |  |    6/8    |          |
|                 |  6/20   |  |  6/10   |  |           |          |
|                  ---------    ---------    -----------           |
|                     ^           | ^            |                 |
|                     |___________| |____________|                 |
|                             DATECHECK                            |
|                                                                  |
|                  ---------    ---------    -----------           |
|                 |PROG1.FTN|->|PROG1.OBJ|->|           |          |
|                 |         |  |         |  |           |          |
|                 |  6/20   |  |  6/20   |  |           |          |
|                  ---------    ---------   |ALLPROG.TSK|    NO    |
|                  ---------    ---------   |           | EXECUTION|
|                 |PROG2.CBL|->|PROG2.OBJ|->|           |          |
|                 |         |  |         |  |           |          |
|                 |  6/20   |  |  6/20   |  |   6/20    |          |
|                  ---------    ---------    -----------           |
|                      |          ^    |           ^               |
|                      |__________|    |___________|               |
|                      COMPILE              LINK                   |
|                                                                  |
 ------------------------------------------------------------------
```

Figure 4-4 COMPLINK Command Functions in the Multi-Module Environment

4.9.4 EDIT Command

The program development language commands load and start the editor for you to create a source or data file. You can also enter the EDIT command to create or modify a source or data file.

Format:

```
EDIT [{ voln:      }] [{ filename        }]
     [{ user voln: }] [{ current program }]
```

Parameters:

voln:	is a 1- to 4-character alphanumeric name specifying the volume on which the source file resides. If this parameter is omitted, the default is the user volume.
filename	is a 1- to 8-character alphanumeric name specifying the file to be created or edited. If this parameter is omitted, the current program is the default.

Functional Details:

A language command entered with a filename loads and starts the editor if the file does not exist. However, if the language command is entered without a filename, enter the EDIT command with a filename to access the editor and create or modify a source file.

If this command is entered in a NULL environment, the tab character is set and displayed, but the language tabs are not set.

If this command is entered with a filename not contained in a multi-module environment, this message is displayed:

```
** FILENAME NOT IN ENVIRONMENT
```

If this command is entered without a filename in the multi-module environment and there is no current program, this message is displayed:

```
   ** CURRENT PROGRAM NOT SPECIFIED
```

If this command is entered without a filename when there is a current program in the multi-module environment, the name of the current program is displayed:

```
     ** EDIT - current program
```

For information on the Edit commands, see Section 1.6.2, or the OS/32 Edit User Guide.

4.9.5 ENV Command

The ENV command entered with an EDF name creates the multi-module environment and allocates the user-specified EDF, if necessary. This command can also be used to clear the current environment.

Multi-Module Format:

$$\text{ENV}\left[\begin{Bmatrix}\text{voln:}\\ \text{user voln:}\\ \text{NULL}\end{Bmatrix}\right]\left[\begin{Bmatrix}\text{filename}\\ \text{current program}\end{Bmatrix}\right]$$

Parameters:

voln:	is a 1- to 4-character alphanumeric name specifying the volume on which the EDF resides. If this parameter is omitted, the default is the user volume.
filename	is a 1- to 8-character alphanumeric name specifying the EDF, filename.EDF. If this parameter is omitted, the default is the current program. The EDF extension is automatically appended and must not be entered by the user.
NULL	clears the current environment.

Functional Details:

If the filename parameter is entered with an extension, this message is displayed:

```
** SYNTAX ERROR
```

If the ENV command is entered without a parameter, the name of the current environment is displayed:

```
** CURRENT ENVIRONMENT = xxxxxxxx
```

If the environment was not set or the NULL parameter was specified, this message is displayed:

```
** NO CURRENT ENVIRONMENT
```

4.9.6 EXEC Command

The EXEC command datechecks source, object, and image modules in language and multi-module environments and compiles or links them if they are outdated. When the image program is current, it is loaded and run.

Format:

$$\text{EXEC}\left[\left\{\begin{matrix}\text{voln:}\\ \text{user voln:}\end{matrix}\right\}\right]\left[\left\{\begin{matrix}\text{filename}\\ \text{current program}\end{matrix}\right\}\right]\left[\text{,"start parameters"}\right]$$

Parameters:

voln:	is a 1- to 4-character alphanumeric name specifying the volume on which the source file resides. If this parameter is omitted, the default is the user volume.
filename	is a 1- to 8-character alphanumeric name specifying the program to be run. If this parameter is omitted, the current program or EDF name is the default.
"start parameters"	are parameters particular to the program to be used. These parameters, usually specified with the operator START command, can now be specified with the program development EXEC command. Start parameters must be entered with beginning and ending quotation marks.

Functional Details:

When the EXEC command is entered in a multi-module environment, all modules contained in the EDF are compiled and linked if they are outdated. The task is then loaded and run.

If start parameters are entered, they are invoked every time the task is executed.

Start parameters must be entered with beginning and ending quotation marks.

EXEC command functions are shown in Figures 4-5 and 4-6.

```
 ---------------------------------------------------------
|      SOURCE, OBJECT, AND IMAGE MODULES BEFORE EXEC      |
|=========================================================|
|                                                         |
|        ---------    ---------    ---------              |
|       |PROG1.FTN|  |PROG1.OBJ|  |PROG1.TSK|             |
|       |         |  |         |  |         |             |
|       |  6/20   |  |  6/18   |  |  6/18   |             |
|        ---------    ---------    ---------              |
|                                                         |
|=========================================================|
|      SOURCE, OBJECT, AND IMAGE MODULES AFTER EXEC       |
|=========================================================|
|                                                         |
|        ---------    ---------    ---------              |
|       |PROG1.FTN|  |PROG1.OBJ|  |PROG1.TSK|             |
|       |         |  |         |  |         |             |
|       |  6/20   |  |  6/18   |  |  6/18   |             |
|        ---------    ---------    ---------              |
|          ^             | ^           |                  |
|          |_____________| |___________|                  |
|                    DATECHECK                            |
|                                                         |
|        ---------    ---------    ---------              |
|       |PROG1.FTN|  |PROG1.OBJ|  |PROG1.TSK|             |
|       |         |->|         |->|         |->   TASK    |
|       |  6/20   |  |  6/20   |  |  6/20   |  EXECUTION |
|        ---------    ---------    ---------              |
|          |            ^ |          ^                    |
|          |____________| |__________|                    |
|            COMPILE          LINK                        |
|                                                         |
 ---------------------------------------------------------
```

Figure 4-5 EXEC Command Functions in the Language Environment

```
 ------------------------------------------------------------------
|           SOURCE, OBJECT, AND IMAGE MODULES BEFORE EXEC          |
|==================================================================|
|                                                                  |
|  ----------    ---------    ---------                            |
| |          |->|PROG1.FTN|  |PROG1.OBJ|                           |
| |          |  |         |  |         |                           |
| |ALLPROG.EDF|  |  6/20   |  |  6/15   |                          |
| |          |   ---------    ---------                            |
| |          |                                                     |
| |          |   ---------    ---------                            |
| |          |->|PROG2.CBL|  |PROG2.OBJ|                           |
| |          |  |         |  |         |                           |
| |          |  |  6/20   |  |  6/15   |                           |
|  ----------    ---------    ---------                            |
|                                                                  |
|==================================================================|
|            SOURCE, OBJECT, AND IMAGE MODULES AFTER EXEC          |
|==================================================================|
|                                                                  |
|              ---------    ---------    ----------                |
|             |PROG1.FTN|  |PROG1.OBJ|  |          |               |
|             |         |  |         |->|          |               |
|             |  6/20   |  |  6/15   |  |          |               |
|              ---------    ---------   |ALLPROG.TSK|              |
|                                       |          |               |
|              ---------    ---------   |          |               |
|             |PROG2.CBL|  |PROG2.OBJ|  |          |               |
|             |         |  |         |->|          |               |
|             |  6/20   |  |  6/15   |  |   6/5    |               |
|              ---------    ---------    ----------                |
|                  ^           | ^            |                    |
|                  |___________| |____________|                    |
|                           DATECHECK                              |
|                                                                  |
|              ---------    ---------    ----------                |
|             |PROG1.FTN|  |PROG1.OBJ|  |          |               |
|             |         |->|         |->|          |               |
|             |  6/20   |  |  6/20   |  |          |               |
|              ---------    ---------   |ALLPROG.TSK|-> TASK       |
|                  |           ^ |      |          |  EXECUTION    |
|              ---------    ---------   |          |               |
|             |PROG2.CBL|  |PROG2.OBJ|  |          |               |
|             |         |->|         |->|          |               |
|             |  6/20   |  |  6/20   |  |   6/20   |               |
|              ---------    ---------    ----------                |
|                  |           ^ |            ^                    |
|                  |___________| |____________|                    |
|                     COMPILE         LINK                         |
|                                                                  |
 ------------------------------------------------------------------
```

Figure 4-6 EXEC Command Functions in the Multi-Module Environment

4.9.7 LINK Command

The LINK command links the object module to yield the image module in language and multi-module environments. If no object module exists, the LINK command compiles the source module to yield the object module. The LINK command does not datecheck, load, nor execute a program.

Language Format:

$$\text{LINK} \left[\begin{Bmatrix} \text{voln:} \\ \text{user voln:} \end{Bmatrix}\right] \left[\begin{Bmatrix} \text{filename} \\ \text{current program} \end{Bmatrix}\right]$$

Multi-Module Format:

```
LINK
```

Parameters:

voln:	is a 1- to 4-character alphanumeric name specifying the volume on which the source file resides. If this parameter is omitted, the default is the user volume.
filename	is a 1- to 8-character alphanumeric name specifying the files to be compiled and/or linked. If this parameter is omitted, the current program is the default. A filename is meaningful only in a language environment.

Functional Details:

When the LINK command is entered in a multi-module environment and no object module exists, all source file fds contained in the EDF are compiled. The resulting object modules are then linked. If a link error occurs, the link sequence aborts, and this message is displayed:

```
** LINK ERRORS:EXECUTION ABORTED
```

If a LINK command is entered when no environment was set, this message is displayed:

```
    ** NO ENVIRONMENT SPECIFIED
```

If there is a compilation error, the process ends with a nonzero end of task code, the link procedure never starts, and the process is aborted. This message is then displayed:

```
    ** COMPILE ERROR - LINK NOT EXECUTED
```

The LINK command also links all of the standard Perkin-Elmer run time libraries specified by the language extension assigned when the source file was created.

4.9.7.1 Link Sequences

The user can specify a link sequence by building a link file that must have the extension .LNK. When the link sequence is specified, the system searches the default user volume for a file with the .LNK extension with a filename matching the EDF name or the current program. When it is found, it is executed.

Example:

```
    *BUILD JOB.LNK
    B>ESTABLISH TASK
    B>INCLUDE PROG1.OBJ
    B>INCLUDE PROG2.OBJ
    B>LIBRARY F7RTL,COBOL.LIB
    B>MAP PR:,AD,AL,XREF
    B>BUILD PROG.TSK
    B>END
    B>ENDB
```

If the user-specified link file is not found, the system uses the default link sequence. There is a default link sequence for each language environment. Following is an example of a default FORTRAN link sequence:

```
    >ESTABLISH TASK
    >INCLUDE current program
    >INCLUDE LIBRARY F7RTL.OBJ/S
    >OP DFLOAT, FLOAT, WORK=X3072
    >BUILD filename.TSK
    >END
```

The LINK command functions are shown in Figures 4-7 and 4-8.

```
 ---------------------------------------------------------
|          SOURCE AND OBJECT MODULES BEFORE LINK          |
|=========================================================|
|                                                         |
|     ---------    ---------                              |
|    |PROG1.FTN|  |PROG1.OBJ|                             |
|    |         |  |         |                             |
|    |   6/20  |  |   6/20  |                             |
|     ---------    ---------                              |
|                                                         |
|=========================================================|
|           SOURCE AND OBJECT MODULES AFTER LINK          |
|=========================================================|
|                                                         |
|     ---------    ---------    ---------                 |
|    |PROG1.FTN|  |PROG1.OBJ|  |PROG1.TSK|                |
|    |         |  |         |->|         |      NO        |
|    |   6/20  |  |   6/20  |  |   6/20  |  EXECUTION     |
|     ---------    ---------    ---------                 |
|                      |             ^                    |
|                      |_____________|                    |
|                           LINK                          |
|                                                         |
|=========================================================|
|                SOURCE PROGRAM BEFORE LINK               |
|=========================================================|
|                                                         |
|     -----------                                         |
|    | PROG1.FTN |                                        |
|    |           |                                        |
|    |    6/20   |                                        |
|     -----------                                         |
|                                                         |
|=========================================================|
|                 SOURCE PROGRAM AFTER LINK               |
|=========================================================|
|                                                         |
|     ---------    ---------    ---------                 |
|    |PROG1.FTN|  |PROG1.OBJ|  |PROG1.TSK|                |
|    |         |->|         |->|         |      NO        |
|    |   6/20  |  |   6/20  |  |   6/20  |  EXECUTION     |
|     ---------    ---------    ---------                 |
|          |           ^             ^                    |
|          |___________|_____________|                    |
|            COMPILE        LINK                          |
|                                                         |
 ---------------------------------------------------------
```

Figure 4-7 LINK Command Functions in the Language Environment

```
 ----------------------------------------------------------------
|            SOURCE AND OBJECT PROGRAMS BEFORE LINK              |
|================================================================|
|                                                                |
|    -----------    ---------    ---------                       |
|   |           |->|PROG1.FTN|  |PROG1.OBJ|                      |
|   |           |  |         |  |         |                      |
|   |           |  |  6/20   |  |  6/20   |                      |
|   |ALLPROG.EDF|   ---------    ---------                       |
|   |           |                                                |
|   |           |   ---------    ---------                       |
|   |           |->|PROG2.CBL|  |PROG2.OBJ|                      |
|   |           |  |         |  |         |                      |
|   |           |  |  6/20   |  |  6/20   |                      |
|    -----------    ---------    ---------                       |
|                                                                |
|================================================================|
|            SOURCE AND OBJECT PROGRAMS AFTER LINK               |
|================================================================|
|                                                                |
|                   ---------    ---------    -----------         |
|                  |PROG1.FTN|  |PROG2.OBJ|  |           |        |
|                  |         |  |         |->|           |        |
|                  |  6/20   |  |  6/20   |  |           |        |
|                   ---------    ---------   |           |        |
|                                            |ALLPROG.TSK|   NO   |
|                   ---------    ---------   |           |EXECUTION|
|                  |PROG2.CBL|  |PROG2.OBJ|  |           |        |
|                  |         |  |         |->|           |        |
|                  |  6/20   |  |  6/20   |  |   6/20    |        |
|                   ---------    ---------    -----------         |
|                                   |             ^               |
|                                   |_____________|               |
|                                        LINK                     |
|================================================================|
|                SOURCE MODULE BEFORE LINK                        |
|================================================================|
|                                                                |
|    -----------    ---------                                    |
|   |           |  |PROG1.FTN|                                   |
|   |           |->|         |                                   |
|   |           |  |  6/20   |                                   |
|   |ALLPROG.EDF|   ---------                                    |
|   |           |                                                |
|   |           |   ---------                                    |
|   |           |->|PROG2.CBL|                                   |
|   |           |  |         |                                   |
|   |           |  |  6/20   |                                   |
|    -----------    ---------                                    |
|                                                                |
|================================================================|
|                  SOURCE MODULE AFTER LINK                       |
|================================================================|
|                                                                |
|                   ---------    ---------    -----------         |
|                  |PROG1.FTN|  |PROG1.OBJ|  |           |        |
|                  |         |  |         |->|           |        |
|                  |  6/20   |  |  6/20   |  |           |        |
|                   ---------    ---------   |           |        |
|                                            |ALLPROG.TSK|   NO   |
|                   ---------    ---------   |           |EXECUTION|
|                  |PROG2.CBL|  |PROG2.OBJ|  |           |        |
|                  |         |  |         |->|           |        |
|                  |  6/20   |  |  6/20   |  |   6/20    |        |
|                   ---------    ---------    -----------         |
|                      |            ^ |           ^               |
|                      |____________| |___________|               |
|                        COMPILE          LINK                    |
|                                                                |
 ----------------------------------------------------------------
```

Figure 4-8 LINK Command Functions in the Multi-Module Environment

4.9.8 LIST Command

The LIST command lists the fds of all the multi-module environment programs that are contained in the current EDF.

Format:

```
LIST
```

Functional Details:

The LIST command causes a listing to be sent to the list device | specified by SSYSPRT when lu assignments were made. When this command is entered, this message is displayed:

```
** CURRENT ENVIRONMENT = current EDF
```

If the LIST command is entered and no fds are in the multi-module environment, the following message is displayed:

```
** ENVIRONMENT EMPTY
```

If an argument is specified with the LIST command, this message is displayed:

```
** TOO MANY ARGUMENTS
```

4.9.9 REMOVE Command

The REMOVE command deletes specified source fds from the current multi-module environment.

Format:

```
REMOVE fd
```

Parameters:

fd	is a file descriptor of a source file contained in the EDF.

Functional Details:

When the REMOVE command is entered, the current EDF is searched for the specified fd. When found, the fd is removed from the multi-module environment. If the fd is not found, the following message is displayed:

```
** FILENAME NOT IN ENVIRONMENT
```

If the fd is omitted or is in an incorrect format, this message is displayed:

```
** SYNTAX ERROR
```

When all of the fds have been removed from the multi-module environment, this message is displayed:

```
** ENVIRONMENT EMPTY
```

4.9.10 RUN Command

The RUN command loads and runs the image program in language and multi-module environments. This command does not datecheck, compile, or link.

Format:

```
    [( voln:      )] [( filename        )] [,"start parameters"]
RUN [{            }] [{                 }]
    [( user voln: )] [( current program )]
```

Parameters:

voln:	is a 1- to 4-character alphanumeric name specifying the volume on which the image module resides. If this parameter is omitted, the default is the user volume.
filename	is a 1- to 8-character name specifying the image module. If this parameter is omitted, the default is the current program.
"start parameters"	are parameters particular to the assembler or compiler being used. These parameters, usually specified with the operator START command, now can be specified with the program development RUN command.

Functional Details:

If a filename is not entered with the RUN command and a task with the filename of the current program does not exist in the language environment, this message is displayed:

```
** fd NONEXISTENT
```

See Section 4.9.6 for more information on start parameters.

Figures 4-9 and 4-10 illustrate the RUN command functions.

```
 ----------------------------
|  IMAGE MODULE BEFORE RUN   |
|============================|
|                            |
|     ---------              |
|    |PROG1.TSK|             |
|    |         |             |
|    |  6/20   |             |
|     ---------              |
|                            |
|============================|
|  IMAGE MODULE AFTER RUN    |
|============================|
|                            |
|     ---------              |
|    |PROG1.TSK|             |
|    |         |->  TASK     |
|    |  6/20   |  EXECUTION  |
|     ---------              |
|       RUN                  |
 ----------------------------
```

Figure 4-9 RUN Command Function in the Language Environment

```
 ------------------------------
|   IMAGE MODULE BEFORE RUN    |
|==============================|
|                              |
|     -----------              |
|    |           |             |
|    |           |             |
|    |           |             |
|    |ALLPROG.TSK|             |
|    |           |             |
|    |           |             |
|    |           |             |
|    |   6/20    |             |
|     -----------              |
|                              |
|==============================|
|   IMAGE MODULE AFTER RUN     |
|==============================|
|                              |
|     -----------              |
|    |           |             |
|    |           |             |
|    |           |             |
|    |           |->   TASK    |
|    |ALLPROG.TSK|  EXECUTION  |
|    |           |             |
|    |           |             |
|    |   6/20    |             |
|     -----------              |
|         RUN                  |
|                              |
 ------------------------------
```

Figure 4-10 RUN Command Function in the Multi-Module Environment

Table 4-4 summarizes the functions of the commands used to compile, link, and run a program.

TABLE 4-4 PROGRAM DEVELOPMENT COMMANDS THAT COMPILE, LINK, AND EXECUTE

COMMAND	FUNCTION
COMPILE	Compiles source module into object module when object module does not exist or is outdated.
COMPLINK	Datechecks source, object, and image modules, and compiles and/or links them if outdated to form image program.
LINK	Compiles source module into object module when object module does not exist. Then links object module and standard run time libraries to form image program.
EXEC	Datechecks image, object, and source modules. Compiles and links them if outdated. Loads and runs up-to-date image program.
RUN	Loads and runs image program without datechecking, compiling, or linking.

4.10 SAMPLE PROGRAM DEVELOPMENT SESSIONS

This section presents coding examples using the program development commands.

```
*FORT TEST
** NEW PROGRAM
-EDIT

   .
   .
   .
(edit session)
   .
   .
   .
SAVE*
>END
```

Create FORTRAN language environment with the FORT language command.

Specify TEST as filename to be allocated. FORT command loads and starts editor with TEST.FTN as current program.

```
*SSYSIN CON:                         Define and set new global
                                     variables.

*SSYSOUT CON:
*SSYSLIST PR:
*EXEC TEST                           Execute TEST.FTN.
-FORTRAN:TEST                        Compile TEST.FTN.
** COMPILE ERRORS, LISTING ON PR:    Compilation errors in TEST.


*EDIT
-EDIT - TEST.FTN                     Find and correct errors.
    .
    .
    .
(edit session)
    .
    .
    .
SAVE*
>END

*EXEC                                Execute current program.
-FORTRAN - TEST
    .
    .
    .
(compilation sequence)               Compile.
    .
    .
    .
-END OF TASK CODE=0                  Sucessful compilation.
-LINK - TEST                         Link the newly created
                                     object module TEST.OBJ.
    .
    .
    .
(link sequence)
    .
    .
    .
-END OF TASK CODE=0                  Successful link. New task
                                     now exists.

** EXECUTION OF TEST FOLLOWS:        Run the new task TEST.TSK.
    .
    .
    .
(execution sequence)
    .
    .
    .
-END OF TASK CODE=0
```

```
      *EXEC                               Successful execution.
                                          Re-execute.
      ** EXECUTION OF TEST FOLLOWS:       Ensure program is compiled
                                          and linked.
         .                                Compile, link unnecessary.
         .                                Object and image up-to-date.
         .
      (execution sequence)
         .
         .
         .
|     -END OF TASK CODE = 0               Successful execution.
|     *RUN

|     ** EXECUTION OF TEST FOLLOWS:       Rerun.
         .
         .
         .
      (execution sequence)
         .
         .
         .
      -END OF TASK CODE=0

      *EXEC NEWPROG                       Execute NEWPROG.
      ** FILE NEWPROG.FTN NOT FOUND       System finds NEWPROG.MAC.
                                          Cannot find NEWPROG.FTN.
                                          Specifiy MACRO command to
|                                         access NEWPROG.MAC and enter
|                                         a new language environment.
      *MACRO

      *EXEC NEWPROG                       Execute NEWPROG.MAC.
      -MACRO - NEWPROG                    Expand.
      -CAL - NEWPROG                      Assemble.
|     -LINK - NEWPROG                     Link.
         .
         .
         .
      (link sequence)
         .
         .
         .
      ** EXECUTION OF NEWPROG FOLLOWS:
         .
         .
         .
      (execution sequence)
         .
         .
         .
      -END OF TASK CODE=0                 Successful execution.
```

```
*EDIT                                  Edit current program.
 EDIT-NEWPROG.MAC
    .
    .
    .
(edit session)
    .
    .
    .
SAVE*
>END

*EXEC                                  Execute current program.
-MACRO - NEWPROG                       Expand.
-CAL - NEWPROG                         Assemble.
-LINK - NEWPROG                        Link.
    .
    .
    .
(link sequence)
    .
    .
    .
** EXECUTION OF NEWPROG FOLLOWS:
    .
    .
    .
(execution sequence)
    .
    .
    .
-END OF TASK CODE=0                    Successful execution.


                                       Create multi-module envi-
                                       ronment with ENV command.

*ENV BIGTASK                           BIGTASK.EDF allocated.
** NEW ENVIRONMENT
*ADD SUB.CAL                           Add 3 module names to EDF.
*ADD MACRTY.CAL
*ADD FTOR.FTN
*LIST                                  List all modules in EDF.
** CURRENT ENVIRONMENT=BIGTASK.EDF
-SUB.CAL
-MACRTY.CAL
-FTOR.FTN
*ADD SUBFUNC.FTN                       Add 2 more modules to EDF.
*ADD YSUB.MAC
```

```
      *REMOVE SUB.CAL                      Remove fd from EDF.
      *FORT SUBFUNC
      -EDIT - SUBFUNC                      Make changes to SUBFUNC.FTN.
          .
          .
          .
      (edit session)
          .
          .
          .
      SAVE*
      >END
|     *EDIT YSUB                           Make changes to YSUB.MAC.
          .
          .
          .
      (edit session)
          .
          .
          .
      SAVE*
|     >END
      *ENV BIGTASK                         Create multi-module envi-
                                           ronment
      *EXEC                                Execute modules remembered
|     -FORTRAN - FTOR.FTN                  in BIGTASK.EDF.
|     -FORTRAN - SUBFUNC.FTN               FTOR.OBJ and YSUB.OBJ
      -MACRO - YSUB.MAC                    modules are outdated.
      -CAL - MACRTY.CAL
      -LINK - BIGTASK                      Link BIGTASK.
          .
          .
          .
      (link sequence)
          .
          .
          .
|     END OF TASK CODE=0                   All objects are linked;
|                                          appropriate RTLs are also linked.

      ** EXECUTION OF BIGTASK FOLLOWS:
          .
          .
          .
      (execution sequence)
          .
          .
          .
      -END OF TASK CODE=2                  Execution errors traced to YSUB.
```

```
*MAC                                  Create language environment.
*EDIT YSUB                            Correct errors in YSUB.MAC.
   .
   .
   .
(edit session)
   .
   .
   .
SAVE*
>END

*ENV BIGTASK                          Enter multi-module environment.
*EXEC
-MACRO:YSUB.MAC                       YSUB.MAC object is outdated.
                                      Expand, assemble, and linkedit.
-CAL - YSUB.MAC
-LINK - BIGTASK
   .
   .
   .
(link sequence)
   .
   .
   .
** EXECUTION OF BIGTASK FOLLOWS:
   .
   .
   .
(execution sequence)
   .
   .
   .
-END OF TASK CODE=0
```

CHAPTER 5
MULTI-TERMINAL MONITOR (MTM) BATCH PROCESSING

5.1 INTRODUCTION

In addition to interactive processing capabilities, MTM also supports concurrent batch processing, allowing the user to run multiple batch jobs from a single batch queue. This feature enables the user to effectively utilize the capabilities of the system with minimal interference to the interactive users.

The number of concurrent batch jobs allowed at any time under MTM is set by the operator from the system console. This number cannot exceed 64. If more batch jobs are submitted than there are active jobstreams, MTM queues the requests until a jobstream becomes available.

The batch queue is an indexed file containing the file descriptor (fd) of the jobs to be processed. Each job is identified in the queue by the fd of the command file. The batch queue is ordered in priority order and in first-in/first-out (FIFO) basis within a priority.

Tasks executing in the batch environment run at a priority lower than or equal to the tasks in the terminal environment. Thus, a batch job executes when the system is not occupied with work from a terminal user. Batch jobs use the processor's idle time and therefore increase the efficiency of the system.

5.2 BATCH COMMANDS

The batch job file consists of a series of MTM user commands and/or command substitution system (CSS) calls. The commands presented in this section are unique to the batch environment.

To submit a batch job a user must have created a batch job file on disk. This file must have a SIGNON command as the first record, and a SIGNOFF command as the last record. The only valid commands to be used between the SIGNON and SIGNOFF commands are MTM user commands (Chapter 2), program development commands (Chapter 4), batch processing conmmands, and calls to a CSS file (Chapter 6). A batch job file is not a CSS. Therefore, CSS commands, with the exception of $IF..., $ELSE, and $ENDC, are invalid. Any command that can be used at a terminal can be used in the batch job file.

Example:

of a single batch job file:

```
SIGNON TEST1,1,PWD
L TEST 1
ST
SIGNOFF
```

Example:

CSS to build a batch job file and submit job:

```
** ASM.CSS     [MODULE]
**
**      @ 1  (MODULE TO BE ASSEMBLED)
**
**      EXAMPLE: ASM EXIN
**
$BU     @1.JOB
SIGNON @1
XAL     @1.LOG,IN,80
LOG     @1.LOG,5
ASM/G   @1
$IFE    0
   MESS LEE *** @1.JOB COMPLETE ***
$ELSE
MESS LEE *** @1.JOB ERROR ***
-$ENDC
SIGNOFF
$ENDB
SUB     @1.JOB,DEL
INQ
$EXIT
```

5.2.1 INQUIRE Command

The INQUIRE command queries the status of a job on the batch queue.

Format:

$$\underline{\text{I}}\text{NQUIRE}\left[\text{fd}\left\{\begin{array}{c},\text{fd}_1\\ \text{user console}\end{array}\right\}\right]$$

Parameters:

fd	identifies the job for which the status is desired. If fd is not specified, all jobs with account numbers the same as the user's are displayed.
fd1	specifies the file or device to which the display is output. If this parameter is omitted, the default is the user console.

Functional Details:

When this command is entered by a privileged user, information about all jobs on the system is displayed. Standard MTM users see just the jobs related to the user's private account. This command can be entered in command mode, task loaded mode, and task executing mode.

Possible responses to the INQUIRE command are:

```
JOB  fd  NOT FOUND

JOB  fd  EXECUTING

JOB  fd  WAITING  BEHIND=n

NO JOBS WITH YOUR ACCOUNT
```

Examples:

INQ	All jobs with the user account number are displayed.
INQUIRE TASK.JOB	The status of TASK.JOB is displayed.

5.2.2 LOG Command

The user can invoke a batch job to produce a log of its commands by including the LOG command and the $COPY command within the batch stream.

Format:

```
                ┌ ┌ ┌ NOCOPY ┐ ┐ ┐   ┌ ┌ n  ┐ ┐
LOG [fd]        │,│ │        │ │ │ , │ │    │ │
                └ └ └ COPY   ┘ ┘ ┘   └ └ 15 ┘ ┘

                ┌ ┌ ┌ NOCOPY ┐ ┐ ┐   ┌ ┌ n  ┐ ┐
SET LOG [fd]    │,│ │        │ │ │ , │ │    │ │
                └ └ └ COPY   ┘ ┘ ┘   └ └ 15 ┘ ┘
```

Parameters:

fd	is the file descriptor of the log file or device. If no fd is specified, logging is terminated. If fd is a file, it must be previously allocated. Files are assigned EWO privileges so that logged output is added to the end of the file. If a log is active when a second LOG command is entered, the old log is closed and the new one is initiated.
COPY	specifies that all output is written to both the terminal and the log device.
NOCOPY	specifies that all output, except messages, is written to the log device and not the terminal. Messages from other users and the operator are written to both the terminal and the log device.
n	is a decimal number from 0 through 65,535 specifying the number of lines after which the log file is to be checkpointed. If this parameter is omitted, the default is 15 lines. If n is specified as 0, no checkpointing occurs.

Functional Details:

The LOG and the SET LOG commands are the same. The command can be entered either way, and both formats perform the same function.

Checkpointing may be done on any type of file. However, on contiguous files, the checkpoint operation is treated as a no-operation. On nonbuffered indexed and extendable contiguous files, the checkpoint operation is useful only if the file is being expanded. On indexed files it is possible that a significant amount of time may elapse between the time the data to be written to the disk leaves the user's buffer and the time that it is physically transferred to the disk. In these cases, checkpointing "flushes" the system buffers, as well as updating the file size in the directory. In general, checkpointing is justifiable only under very specific circumstances, such as when a very large amount of data is written to an indexed file over an extended period of time, without the file being closed.

Example:

```
LOG PR:
```

5.2.3 PURGE Command

The PURGE command purges a submitted job from the batch queue.

Format:

```
    PURGE fd
```

Parameter:

fd	is the file descriptor of the job to be purged. Only jobs with the user account number can be purged.

Functional Details:

If the specified job is executing, it will be cancelled or terminated. If the job is waiting to be run it will be removed from the batch queue.

Example:

```
    PURGE  TASK.JOB         TASK.JOB  is purged.
```

5.2.4 SIGNOFF Command

The last command in a batch stream must be the SIGNOFF command.

Format:

```
    SIGNOFF
```

Functional Details:

When a terminal user signs off the system, these messages are displayed:

```
    ELAPSED TIME=hh:mm:ss          CPUTIME=utime/ostime
    SIGNON LEFT=hh:mm:ss           CPU LEFT=hh:mm:ss
    TIME OFF=mm/dd/yy  hh:mm:ss
```

The SIGNOFF command can be entered in command mode, task loaded mode, and task executing mode.

5.2.5 SIGNON Command

SIGNON must be the first command in a batch job.

Format:

```
                                        ┌                   ┌fd       ┐┐
SIGNON userid ,actno,password           │,ENVIRONMENT=      │         ││
                                        │                   └NULL [:] ┘│
          [,CPUTIME=maxtime]            └                              ┘
          [,classid=iocount1 [,...,classid=iocount32]]
```

Parameters:

userid	is a 1- to 8-character alphanumeric string specifying terminal user identification.
actno	is a 5-digit decimal number specifying the terminal user's account number. This must be a valid account number in the AUF file and can never exceed 65,535. If this parameter is omitted, the password parameter should also be omitted. MTM will use the account number of the user submitting the batch job.
password	is a 1- to 12-character alphanumeric string specifying the terminal user's password. This parameter should be omitted if the actno parameter is omitted. MTM will use the password of the user submitting the job.
ENVIRONMENT=	fd is the file descriptor specifying the file that will establish the user's environment at signon time.
	NULL specifies that the signon CSS procedure, USERINIT.CSS, should be ignored and the user will establish the environment at signon time. If the entire keyword parameter is omitted, MTM searches all online disks for the signon CSS procedure, USERINIT.CSS/P. The system volume, system account, is searched last. If USERINIT.CSS is found, MTM calls the CSS and executes the routine. If it is not found, MTM enters command mode.

CPUTIME=	maxtime is a decimal number specifying the maximum CPU time to which the batch job is limited. If this parameter is omitted, the default established at sysgen is used. If 0 is specified, no limits are applied. The parameter can be specified as: mmmm:ss hhhh:mm:ss ssss
classid=	is one of the 4-character alphanumeric mnemonics, specified at sysgen, associated with each specified device or file class.
iocount	is a decimal number specifying the maximum number of I/O transfers associated with a particular device class to which the batch job is limited. If this parameter is omitted, the default established at sysgen is used. If 0 is specified, no limits are applied to that class.

Functional Details:

Between the SIGNON and SIGNOFF commands, any command or CSS call that is valid from the terminal is allowed. A SIGNON command cannot be followed by another command, on the same line, separated by semicolons. When ENVIRONMENT=NULL is specified, the colon is optional. This allows the user the ability to specify the null device (NULL:).

The account number and password can be omitted if a batch job is submitted from a user terminal. If a batch job is submitted from the system console or via the Spooler, the account number and password must be specified.

| The ENVIRONMENT= parameter may be ignored, depending on the user account's privileges.

Examples:

```
SIGNON  ME

S  ME,12,PSWD,CPUTIME=2:30:00,DEV1=150

S  ME,CPUTIME=120

S  ME,ENV=NULL,CPUTIME=120

S  ME,ENV=XYZ
```

5.2.6 SUBMIT Command

The terminal user adds a job to the batch queue with the SUBMIT command.

Format:

```
    SUBMIT fd [,DELETE] [,PRIORITY=priority]
```

Parameters:

fd	is the file descriptor of the file submitted to batch.
DELETE	deletes the batch job file created to submit the batch job. If this parameter is omitted, the batch job file remains on the user volume.
PRIORITY=	priority is a decimal number which specifies the priority at which a batch job will run. The range of valid priority numbers is dependant upon the user's account privileges, sysgen options, and MTM's priority. The maximum range allowable is MTM's priority + 1 through 255. If this parameter is omitted, a batch job will run at the default batch priority (the default batch priority is 12 lower than MTM's priority plus the value specified at MTM sysgen time for batch priority) or the Link priority (the priority established when the task was built), whichever is lower.

Functional Details:

The priority at which a batch job runs is relative to MTM's priority and the default batch priority established at MTM sysgen time. The u-task priorities are established at link time and can be reset with the PRIORITY parameter of the SUBMIT command. Interactive tasks run at the default priority of 12 priorities lower than MTM. Batch jobs run at the default priority of 12 lower than MTM plus the value specified at MTM sysgen time. If the MTM sysgen priority is set to equal 1 and MTM's priority equals 128, interactive jobs will run at priority +12 (140), or 12 lower than MTM; batch jobs will run at priority +13 (141), 13 lower than MTM.

The rules for establishing priorities are:

- Batch jobs can run at the same priority as interactive tasks but not higher than interactive tasks if the user account has this privilege enabled; otherwise they are run at (maximum) one priority lower than interactive tasks.

- If a valid priority is specified, the batch job runs at that priority or the link priority, whichever is lower.

- If the specified priority is invalid, the default priority is assigned by MTM, and the following message is displayed:

```
WARNING - REQUESTED PRIORITY n ILLEGAL, n USED
```

- If the specified priority is greater than 255, 255 is used.

- If no u-task priority is specified with the SUBMIT command, the batch job runs at the default priority or the link priority, whichever is lower.

The SUBMIT command can be entered in command mode, task loaded mode, and task executing mode.

Example:

Create a batch job stream from the terminal via the BUILD...ENDB sequence:

```
BUILD TEST.JOB
SIGNON ME,ENV=NULL
LOG PR:
L TEST.TSK
AS 3,PR:
START
SIGNOF
ENDB
```

Submit the job from the terminal for batch processing:

```
SUBMIT TEST.JOB
```

Submit a batch job file and have it deleted after the batch job execution is complete:

```
SUBMIT XYZ.JOB, DELETE
```

Submit a batch job and have it run at the same priority as an interactive job:

```
SUBMIT XYZ.JOB, P=129
```

5.3 BATCH JOB SUBMISSION USING THE SPOOLER

The Spooler is also used to submit batch jobs to the batch queue for execution under MTM. Batch jobs submitted through the Spooler later can be resubmitted as a batch job through the terminal.

5.4 ERROR HANDLING

Any error that occurs in a batch job file causes automatic termination of the job, and a message is written to the log file or device. If a batch task pauses, the task is cancelled by MTM with an end of task code of 255, and the job is terminated, unless the batch task pause option was enabled at MTM sysgen. See Section 5.5. When a batch task completes, the end of task code can be tested by subsequent commands in the batch stream to determine if the task completed normally.

5.5 BATCH TASK PAUSE OPTION

This option allows a batch task to pause without being cancelled immediately by MTM. MTM logs the following message to the system console if a batch task enters the paused state:

```
hh:nn:SS     .MTM > task id     BTCH TSK PAUSED
```

In this message task id is the name of the batch task that has paused. The system operator has the option to cancel or continue the paused batch task.

5.6 EFFECT OF RESTRICTED DISKS ON BATCH JOBS

When accounts with access to restricted disks are given read/write access, batch jobs are not affected. If read-only or no access is specified, messages are not displayed on the user console. If a submit file for a batch job is on a restricted disk and account 0 does not have read/write access, the following message is displayed on the system console:

```
.MTM:BATCH ASGN-ERR TYPE=PRIV JOB=fd
```

CHAPTER 6
COMMAND SUBSTITUTION SYSTEM (CSS)

6.1 GENERAL DESCRIPTION

The command substitution system (CSS) is an extension to the OS/32 command language enabling the user to establish files of dynamically modifiable commands that can be called from the terminal or other CSS files and executed in a predefined sequence. In this way, complex operations can be carried out by the terminal user within only a few commands. CSS provides:

- the ability to switch the command input stream to a file or device,
- a set of logical operators to control the precise sequence of commands,
- the ability to pass both positional parameters and keyword parameters to a CSS file so that general sequences take on specific meaning when the parameters are substituted or the keyword encountered in the CSS,
- the ability to specify replacement characters within a CSS line to alter the function of the line when executed,
- the ability to perform decimal and hexadecimal computation and conversion within a CSS line (addition, subtraction, multiplication, and division),
- the ability to use standard local and global variables or new global and new internal variables that introduce extended power and flexibility to variable usage within a CSS,
- the ability to perform searches within specified CSS calls to subtract specific sections of the call and use them as replacements within the CSS, and
- the ability for one CSS file to call another, in the manner of a subroutine, so complex command sequences can be developed.

A CSS file is simply a sequential text file. It can be a deck of cards, a magnetic tape, or a disk file. An example of a simple CSS file is:

```
*THIS IS AN EXAMPLE OF A CSS FILE
LOAD TEST.TSK/G,5
ALLOCATE XXXDIX.DTA,CO,40
AS 1,INPUT.DTA
AS 2,XXXDIX.DTA;AS 5, CON:
ASSIGN 3,PRT:;*LU3-LINEPRINTER
START
$EXIT
```

NOTE

Blank lines are ignored. The semicolon allows more than one command to be entered on the same line. Null CSS commands (;;) are ignored. An asterisk introduces a comment.

6.2 CALLING A CSS FILE

A CSS file is called and executed from the terminal by specifying the file descriptor (fd) of the CSS file. If only the filename is specified, MTM appends the extension.CSS and first searches the user default volume in the user's private account. If the file is not found, the system volume system account is searched. If the volume name or account class is specified by the user, a system default will not be tried. A user must have the CSS privilege in order to call CSS files in the user's private account or group. If not privileged he may only call system CSS's. If the user also has the privilege to specify account numbers instead of classes, he may call a CSS in any account. If the leading characters of a CSS fd are the same as a command, MTM assumes a command:

Example:

CLO.CSS	CLOSE	MTM assumes the CLOSE command.
AS3.CSS	ASSIGN 3	MTM assumes the ASSIGN command.

By specifying a volume name and/or extension, a CSS file that otherwise would conflict with an MTM command can be called.

Example:

```
M300:CLOSE
M300:CLOSE.CSS
```

6.3 USE OF PARAMETERS

The CSS call can have parameters. The parameters are entered after the CSS fd and are separated from it by one character space. If there is more than one parameter, each is separated by commas. If a parameter contains the double quote character ("), or single quote character (') all parameters up to the next double quote character are passed as one parameter. Null parameters are permitted.

Example:

ABC P1, "P2A, P2B" calls CSS file ABC.CSS on the default volume with two parameters. Parameter 1 is P1. Parameter 2 is P2A, P2B.

JUMP ,,C calls CSS file JUMP.CSS on the default volume with three parameters; the first two are null.

Within a CSS file, a parameter is referenced by the use of the special symbol "@n" where n is a decimal integer number indicating which parameter the user is referencing. Parameters are numbered starting with 1. Parameter 0 has special meaning and is defined later in this section. The first parameter is referenced by @1, the second @2, etc. A straightforward text substitution is employed.

Example:

A CSS file ROG consists of:

```
LOAD     @1
START    @3,@2
```

It is called as follows:

```
ROG PROGRAM,NOLIST,148
```

Before each line of the CSS file is decoded, it is preprocessed, and any reference to a parameter is substituted with the corresponding text. Thus, the file ROG with the previous call is executed as:

```
LOAD PROGRAM
START 148,NOLIST
```

| @1 is replaced with PROGRAM (the 1st parameter in the CSS call).
| @3 is replaced with 148 (the 3rd parameter in the CSS call).
| @2 is replaced with NOLIST (the 2nd parameter in the CSS call).

Example:

All of the following references to Parameter 12 are valid expressions:

```
@12 or @12ABC or @12.EXT
```

This mechanism allows concatenation. For instance, if the first command in file ROG were LOAD @1.TSK, only those files with the extension .TSK would be presented to the loader. Concatenation of numbers requires care. 123@1 references Parameter 1, but @1123 is a reference to Parameter 1123. A reference to a nonexistent parameter is null.

The multiple @ facility enables a CSS file to access parameters of higher level files. CSS files can call each other to a maximum depth specified at sysgen time. Thus, @@2 in a CSS file refers to the second parameter of the calling file.

Example:

Given the CSS call:

```
CSS1 arg1,arg2
```

and assuming that in file CSS1 there is another call:

```
CSS2 arg3,arg4
```

the following references can be made in CSS2:

```
@1  = arg3
@2  = arg4
@@1 = arg1
@@2 = arg2
```

If a multiple @ sequence is such that the calling level referred to is nonexistent, the parameter is null.

Parameter @0 is a special parameter used to reference the name of the CSS file in which it is contained. Parameter @0 is replaced during the preprocessing of the command line with precisely the same fd used to call the file.

Example:

A CSS file consists of:

```
AS 1,@0
$EXIT
```

If this file is called from the card reader (CR:), then lu1 is assigned to the card reader (CR:). Likewise, a call from the magnetic tape (MAG1:) results in:

```
AS 1,MAG1:
```

6.4 USE OF KEYWORDS

In the previous section the usage of positional parameters was presented. The CSS language also provides a means of passing keywords in a CSS call. Again a straightforward substitution procedure is applied. Keywords enable the user to explicitly specify a value that is subsequently substituted for each reference of the keyword encountered within the CSS. The value of a keyword is defined in the CSS call in the following format:

Format:

```
keyword = [parameter]
```

Parameters:

keyword	is the 1- to 8-character name of a keyword. The characters must be alphabetic (A-Z).
=	is a required delimiter between a keyword and its assigned value for the CSS call. This delimiter must immediately follow the keyword (no blanks allowed).
parameter	is a character string which replaces the keyword reference with the CSS. Null parameters are allowed.

Functional Details:

The following rules apply for the use of keywords within a CSS file and the relationships between keywords and positional parameters.

- The leading blanks of a keyword parameter are skipped unless they are included with the parameter through the use of single ('...') or double ("...") quotes.
- All characters between single or double quotes belong to the same parameter. This allows the user to define a parameter with leading blanks, semicolons, commas, or an equal sign. A carriage return is not allowed within the parameter definition.
- An equal sign (=) (by default) marks the keyword. This equal sign can be altered (via the SET KEYOPERATOR command) to one of six other characters. Therefore, if you want to define a parameter with an equal sign in it the equal sign must be delimited by single or double quotes or the key operator must be changed to a character other than the equal sign.
- A keyword must never be followed by a positional parameter. All positional parameters must be passed first in the CSS call, then all keywords may follow. Positional parameters and keywords must be separated with commas.

Examples:

These are valid examples of CSS calls using positional parameters and keywords:

```
    TEST ABC.FTN,,BA,OP=BATCH,LI=CON:
    TEST SOURCE=ABC.FTN,LI=CON:
```

These are examples of illegal CSS calls using positional parameters and keywords:

ILLEGAL CSS CALLS	REASON
TEST A,B,FTNOPTION=HOLL	keyword is greater than 8 characters
TEST A,B,OP=HOLL,D	positional parameter D is after a keyword
TEST B,,OP=LNCT=60	double equal signs not valid
TEST B,,=HOLL	keyword name missing
TEST A'='B,C'=D	second quote not matched

6.4.1. Referencing Keywords Within the CSS

Within a CSS file, a keyword parameter is referenced by the use of the @= symbol (similar to the @ symbol usage for positional parameters.

Format:

$\left[@\left[@...@\right]\right]$ @=/[keyword]/

Parameters:

@=	is the symbol which notifies the preprocessor that a reference to a keyword parameter is being made. The use of additional @ symbols is allowed to access keywords of a higher level CSS (same as with positional parameters).
keyword	is a 1- to 8-character keyword (excluding period). The user has the option to define a minimum set of required characters for a keyword. This is accomplished by separating the required characters and the optional characters with a period. Required characters precede the period; optional characters follow the period.

For example, defining a keyword in the following manner:

```
@=/OP.TION/
```

indicates

- the keyword is OPTION, and
- the minimum required character set to reference OPTION is OP.

Functional Details:

If the same keyword mnemonic is passed more than once in a CSS call, the first keyword match found is used in substitution (scanning from left to right in the call).

References to non-existing keywords or to higher CSS levels which do not exist are not expanded, as well as references without a keyword. References with a keyword expand in the usual manner. The following examples show the result of putting keyword references in a CSS file and then passing keyword parameters in the CSS call.

Examples:

CSS file with keyword references:

```
BUILD TEST
$WR [@=/OP.TION/][@=/LI.ST/][@=//]
$EX
ENDB
```

Some calls to the CSS TEST and the results of these calls:

```
        CALL                                RESULT

TEST                                 > [] [] []
TEST A, LIST=PR:                     > [] [PR:] []
TEST OPTION=AA,OP=BB,LI=CON:         > [AA] [CON:] []
TEST A, OP=LNCT'='62,LIST="AB'@'"    [LNCT=62][AB'@'] []
```

Note that in example 3 the first keyword definition for OPTION (AA) is used even though OP=BB is specified. Note also in example 4 that an equal sign can be passed as part of the keyword value as long as it is bracketed with single or double quotes. Single quotes can also be passed as part of the keyword value as long as they are bracketed by double quotes and vice-versa.

6.5 USE OF VARIABLES

MTM and batch users can allocate a specified number of variables to be used within a CSS. In general there are two types of variables, variables that exist from signon to signoff and variables that only exist within a particular CSS level while the CSS is active. There are now further distinctions between the types of variables available with MTM.

6.5.1 Types of Variables

There are now four types of variables within MTM:

- Global variables
- Local variables
- New global variables
- New internal variables

The first two types - global and local variables - should be familiar to all users of previous releases of MTM. Global variables exist from signon to signoff or until they are freed via the $FREE command. Local variables can be used only within the CSS levels in which they are defined. When a particular CSS level is exited, all local variables defined within it are freed.

The maximum number of global and local variables that can be defined is established at MTM sysgen time. See the OS/32 Multi-Terminal Monitor (MTM) System Planning and Operator Reference Manual.

The third and fourth variable types - new global and new internal - are new with this release of MTM. These variables are similar to the local and global variables in terms of usage. However, the way in which they are defined, released, and the capabilities available when defining these variables make them much more powerful and flexible than the previous variables.

New global variables exist from signon through signoff or until they are released via the $RELEASE command or if defined by the $DEFINE command as an undefined value. The number of new global variables allowed in a system is determined at MTM sysgen (maximum of 99). No new global variables are allowed in the system if the new global option is disabled at MTM sysgen.

New internal variables exist only within the CSS level in which they are defined. New internal variables are released automatically on return to the console level. The user may release new internal variables via the $RELEASE command or by using an undefined value via a $DEFINE command. The maximum number of new internal variables that can be used is set at MTM sysgen time. The maximum/limit allowed is 99.

NOTE

Users should familiarize themselves with usage of both new global and new internal variables. These variable types will eventually replace the local or global variables usage. Local and global variable support will eventually be phased out in future releases.

6.5.2 Naming Local or Global Variables

A local or global variable name can consist of 1-to 8-characters and must be preceded by the commercial @ sign. The character following the @ sign must be alphabetic (A-Z); the remaining characters can be alphanumeric.

Examples:

```
    @A
    @B19
    @ABCD1234
```

Local variables are named via the $LOCAL command. GLOBAL variables are named via the $GLOBAL command.

6.5.3 Naming New Global or New Internal Variables

A new global or new internal variable name can consist of 1- to 8-characters. The first character must be alphabetic, the remaining characters can be alphanumeric.

Examples:

```
GD
SS12
S1234567
```

New global and new internal variables are named via the $DEFINE command and at that time are associated with a decimal number. The variable can then be referenced by name or number within a CSS. The following conventions apply to the expansion of a new global or new internal variable within a CSS:

To reference the value of a new global or new internal variable, the following formats can be used;

$$@* \begin{bmatrix} G \\ I \end{bmatrix} \begin{bmatrix} n \\ /name/ \end{bmatrix}$$

Where:

G	specifies a reference to a new global variable
I	specifies a reference to a new internal variable. This is the default.
n	specifies the number of the variable to be referenced
name	specifies the name of the variable

To obtain the name of a new variable use the following format:

```
     ┌ ┐
     │G│
  @* │ │Nn
     │I│
     └ ┘
```

Where:

G	specifies a new global variable.
I	specifies a new internal variable.
n	specifies the number of the variable whose name is being requested.

Examples:

@*G3	references global variable number 3.
@*/VOLUME/	references the internal variable name VOLUME.
@*N3	references the name of internal variable number 3.

6.5.4 CSS Line Expansion

The MTM preprocessor expands the entire CSS line in one step. Because of this, the user is advised to be careful when using the new global or new internal variable name/value in the CSS line after redefining them with a $DEFINE command.

The following illustrates how the preprocessor handles these occurrences:

```
    $DEFINE1,,ST(ORIGINAL)
    $DEFINE1,,ST(NEW);$DEFINE3,,ST(@*1)
```

This expands to:

```
    $DEFINE1,,ST(NEW);$DEFINE3,,ST(ORIGINAL)
```

The value of the new internal variable 3 is not the expected string NEW, but the string ORIGINAL.

6.5.5 Reserved Variables

Variable names starting with the character string @SYS are reserved for system use. A user cannot define variables starting with @SYS. However, a user does have read and write access to @SYS variables.

The global variable @SYSCODE is reserved and contains the value of the last end of task code for a particular session.

6.6 COMMANDS EXECUTABLE WITHIN A CSS FILE

All of the MTM supported commands can be used in a CSS file, as well as a number of commands specifically associated with the CSS facility.

Most of the CSS commands start with the $ character with the exception of the SET CODE and PRIOR commands.

The CSS commands entered within a CSS file are described in the following sections. Refer to Appendix E for CSS message descriptions.

NOTE

If a task is started when CSS is running, CSS becomes dormant until the task is terminated. Execution of the CSS stream will resume after the task terminates.

```
 --------------
|    %...%     |
 --------------
```

6.6.1 Character Replacement Command %...%

The character replacement command (%...%) enables a user to define and replace up to four different characters within a specified CSS line. The user must indicate the line in which replacement is to occur, the new characters, and the characters to be replaced. Unless otherwise specified, every occurrence of a specified character within the line will be replaced.

Format:

$$\%\left\{\begin{array}{l}\text{char1char2}_1\ \left[\text{char1char2}_2\ldots\text{char1char2}_4\right]\ \%\\ \%\ \text{new delimiter}\end{array}\right\}$$

Parameters:

%	is the initial current replacement string delimiter. This indicates the start of the character replacement specification.
char1char2_1 ...char1char2_4	is the specification of the character to be replaced (char1) and the character to be used as the replacement (char2). Up to four of these replacement specifications can be specified. The preprocessor translates this statement as: replace the character specified by char1 with the character specified by char2. If more than one replacement specification is present there must be no blanks between them. If char1 and char2 are the same, char1 is deleted from the CSS line.
% new delimiter	this indicates that a new replacement delimiter (by default the % sign) follows. The new delimiter is the first character after the % sign and is active for the remainder of the CSS line (or until a new delimiter is specified).

Functional Details:

Character replacement operations are only performed in lines which have a percent sign (%) in column 1 of the line. This percent sign (%) is not part of the character replacement command, it merely flags lines eligible for character replacement.

Character replacement is only allowed within a CSS.

The only legal use of blanks within the character replacement delimiters is as replacement characters. The initial replacement delimiter is always reset to % at the beginning of each CSS line and previous replacement characters are deleted. In effect, each CSS line with replacement information is treated as a single entity.

Each usage of the character replacement command resets all previously defined replacement characters. When a new replacement delimiter is specified, all other replacement strings are cleared. The $COPY command suppresses the display or printing of replacement string delimiters and replacement strings.

NOTE

Replacing a character with an @ symbol will result in an additional preprocessing step for that line in order to expand the @ symbol with the appropriate substitution parameter if possible.

The following examples are used to illustrate the basic functionality of the character replacement command. Obviously, the uses of this command are not limited to those shown below. The command becomes extremely powerful as the user introduces more involved substitution and replacement within the same line.

CHARACTER REPLACEMENT CSS LINE	INTERPRETATION	RESULT AFTER PROCESSING
%LO %',%F7D'20	Replace the single quote character (') with the comma (,) in the string F7D'20.	>LO F7D,20
%LO %%\\',\F7D'20	Change the replacement delimiter from % to the \, and replace the single quote character (') with the comma (,) in the string F7D'20.	>LO F7D,20
%LO F7D%',A2B0%'AB	Replace the single quote character (') with a comma (,) replace A with 2, replace B with a 0 in the character string 'AB. The string F7D remains unchanged.	>LO F7D,20
%LO %',%F7D'20;%%%$W'A',	Replace the single quote character with the comma character in the string F7D'20. Then reset the line (clear all replacement instructions for the balance of the line). Because of this the single quotes around A are not replaced.	>LO F7D,20;$W'A'

Another use of the character replacement command is the combination of character replacement and parameter substitution.

Example:

```
    $BUILD TEST

    %%%\\*@\%%+@%$WR @1
    $EX
    $ENDB
```

This example will result in three preprocessing passes through the line in order to complete the requested functions. A step by step analysis will show this. Assume TEST CSS is called with the following call:

```
    TEST *2,+3,'3RD USED'
```

The first preprocessing pass through the line causes the command delimiter to be changed from % to \, the first parameter in the CSS call (*2) replaces the @ 1 reference in the CSS, and the * is replaced with an @ symbol. The line now looks like this:

```
    %%+@%$WR @2
```

The replace to an @ sign requires a second preprocessor pass through the line in order to expand the reference. On the second preprocessing pass through the line, the second parameter in the CSS call (+3) replaces the @2 reference in the CSS line, and then the + is replaced by an @ symbol according to the second character replacement specificaion. The line now looks like this:

```
$WR @3
```

The replace to an @ sign reference requires a third preprocessor pass through the line in order to expand the parameter reference. On this pass the third parameter in the CSS call (3RD USED) is substituted for the @3 reference within the CSS. The line now looks like this:

```
$WR 3RD USED
```

No further preprocessing of the line is required. The final output of this CSS when called as detailed previously would be:

```
-3RD USED
```

6.6.2 $BUILD and $ENDB Commands

The $BUILD command causes succeeding lines to be copied to a specified file up to, but excluding, the corresponding $ENDB command. Before each line is copied, parameter substitution is performed.

Format:

$$\text{\$BUILD}\begin{Bmatrix}\text{fd}\\ \text{lu}\end{Bmatrix}[\text{,APPEND}]$$

```
    .
    .
    .
$ENDB
```

Parameters:

fd	is the output file. If fd does not exist, an indexed file is allocated with a logical record length equal to the command buffer length. If the fd specified does not contain an extension, .CSS is the default. If a blank extension is desired, the period following the filename must be specified.
lu	specifies that a temporary file is to be created and the $BUILD data is copied to it. When $ENDB is encountered, the file is assigned to the specified logical unit of the loaded task. The lu option is valid only when a task is loaded.
APPEND	allows the user to add data to an existing fd. If the fd does not exist, it is allocated.

Functional Details:

The $BUILD command must be the last command on its input line. Any further information on the line is treated as a comment and is not copied to the file.

The $ENDB command must be the first command in the command line, but it need not start in column 1. Other commands can follow $ENDB on the command line, but nesting of $BUILD and $ENDB is not permitted.

6.6.3 $CLEAR Command

The $CLEAR command terminates a CSS stream, closes all CSS files, and deactivates CSS.

Format:

```
    $CLEAR
```

Functional Details:

The $CLEAR command can be entered in command mode, task loaded mode, and task executing mode.

6.6.4 $CONTINUE Command

The $CONTINUE command resumes execution of a CSS procedure suspended by a $PAUSE or $WAIT command.

Format:

```
    $CONTINUE
```

6.6.5 $COPY and $NOCOPY Commands

The $COPY and $NOCOPY commands control the listing of CSS commands on the terminal or log device (if from batch). $COPY initiates the listing and all subsequent commands are copied to the terminal before being executed. The $NOCOPY command deactivates the listing, but is itself listed. The $COPY command is an aid in debugging CSS job streams.

Format:

```
    $COPY

    $NOCOPY
```

6.6.6 $DEFINE Command

The $DEFINE command is used to define or to redefine new global or new internal variables.

Format:

$$\underline{\text{\$DEF}}\text{INE} \left\{ \begin{matrix} \underline{\text{G}}\text{VARIABLE} \\ \text{IVARIABLE} \end{matrix} \text{ n} \right\}, [\text{name}], \text{operator}_1 \ [\text{operator}_2 \ldots \text{operator}_n]$$

Parameters:

GVARIABLE	specifies that a new global variable is being defined. (not allowed if new global option is set off at MTM sysgen).
IVARIABLE	specifies that a new internal variable is being defined. This is the default.
n	is the new variable number. The allowed range is between 1 and the maximum value set at MTM sysgen.
name	is the new global variable or new internal variable name. It is 1- to 8-characters long and can consist of any character A - Z or any number 0 - 9.
operator_1 $\text{operator}_2 \ldots \text{operator}_n$	is one or more of the following operators which selects a particular function to be performed to determine the variable's value.

File Descriptor Operators

ACCOUNT
FILENAME
EXTENSION
VOLUMENAME

Logical Operators

LOGICAL GO
LOGICAL LD
LOGICAL LU
LOGICAL TD
LOGICAL TU

Computation and Conversion Operators

DCOMPUTE
DHCONVERT
HCOMPUTE
HDCONVERT

Other Operators

CLEAR
CURRENT
DVOLUMENAME
REQUIRED
SEARCH
STRING

The following sections define the format and function of each of these operators within the $DEFINE command.

6.6.6.1 File Descriptor Operators

The following four operators can be used to determine the account, filename, extension, or volumename of a specified file descriptor and then assign the determined portion of the fd as the value of the variable being defined.

6.6.6.1.1 ACCOUNT Operator

The ACCOUNT operator of the $DEFINE command enables a user to determine the account designator of a specified file descriptor and assign the designator as the value of the variable being defined.

Format:

$$\text{ACCOUNT}\left(\begin{Bmatrix} \text{fd} \\ = \end{Bmatrix}\right)$$

Parameters:

fd	is the file descriptor of the file or device for which the account designator is to be determined.
=	specifies that the current total result for this $DEFINE command is used to determine the account designator.

Functional Details:

The value returned is /P, /G, or /S depending upon the specified account. If no account is specified, /P is returned for filenames and undefined is returned for devices. If the user has the account number privilege, the account number, rather than an account class, is returned.

Example:

The following CSS is built:

```
$BUILD TEST
$DEFINE 6,,ACCOUNT (@1)
$WR @*6
$EX
$ENDB
```

The above CSS is called with the following call:

```
TEST ABC.FTN/G
```

The result of $WR @*6 is:

```
/G
```

6.6.6.1.2 EXTENSION Operator

The EXTENSION operator of the $DEFINE command enables the user to determine the extension of a given file descriptor and return that extension as the value of the variable being defined.

Format:

$$\text{EXTENSION}\left(\begin{Bmatrix} \text{fd} \\ = \end{Bmatrix}\right)$$

Parameters:

fd	is the file descriptor of the file or device for which the extension is to be determined.
=	the current total result for this $DEFINE command is used to determine the extension.

Functional Details:

The returned value will contain a leading period if an extension was specified, otherwise the value of the variable is undefined.

Example:

```
    BUILD TEST
    $DEFINE 10,,EXTENSION(@1)
    $WR @*10
    $EX
    ENDB
```

When called with the following CSS call:

```
    TEST FORTRAN.FTN
```

The $WR @*10 would output .FTN

6.6.6.1.3 FILENAME Operator

The FILENAME operator of the $DEFINE command enables the user to determine the filename of a given file descriptor and return that value as the defined variable.

Format:

$$\underline{\text{F}}\text{ILENAME}\left(\begin{Bmatrix}\text{fd}\\ =\end{Bmatrix}\right)$$

Parameters:

fd	is the file descriptor of the file or device for which the filename is to be determined.
=	the current total result for this $DEFINE command is used to determine the filename.

Functional Details:

If a file descriptor was specified in the FILENAME operator, the returned value is the filename.

If a device name was specified in the FILENAME operator, the returned value is undefined.

Example:

```
BUILD TEST
$DEFINE 10,,FILENAME(@1)
$WR @*10
$EXIT
ENDB
```

When called with the following CSS call:

```
TEST M301:TCHFIN12.FTN
```

the $WR @*10 result is TCHFIN12

6.6.6.1.4 VOLUMENAME Operator

The VOLUMENAME operator of the $DEFINE command enables the user to determine the volume name of a given file descriptor and assign that value to the variable being defined.

Format:

$$\underline{\text{V}}\text{OLUMENAME}\left(\begin{Bmatrix}\text{fd}\\ =\end{Bmatrix}\right)$$

Parameters:

fd	is a file descriptor of the file for which the volumename is to be determined.
=	the current total result for this $DEFINE is used to determine the volumename.

Functional Details:

The new variable value returned is the specified volume name, or the user's private volume name under MTM. The volume name is always followed by a colon (:).

Example:

```
BUILD TEST.CSS
$DEFINE 20,, VOLUMENAME (M301:SOURCE.FTN)
$WR @*20
$EX
ENDB
```

Calling the above CSS with the following call:

```
*TEST
```

The value of $WR @*20 is:

```
-M301:
```

6.6.6.2 LOGICAL Operators

The LOGICAL operators of the $DEFINE command enable the user to test the current or last result as defined, exit from the $DEFINE command, or skip operators within the $DEFINE command.

Format:

$\underline{\text{GO}}\text{n}$

$$\text{L}\begin{Bmatrix}\text{D}\\\text{U}\end{Bmatrix}\begin{Bmatrix}\text{n}\\\text{\$name}\end{Bmatrix}$$

$$\text{T}\begin{Bmatrix}\text{D}\\\text{U}\end{Bmatrix}\begin{Bmatrix}\text{n}\\\text{\$name}\end{Bmatrix}$$

Parameters:

GO	specifies an unconditional skip of operators or an exit from within the $DEFINE command.
n	is a decimal number between 0 and 999. Where: 0 = exit the $DEFINE command. 1-999 = skip this number of operators
L	specifies that the result of the last operator is to be tested. The test performed depends upon whether the D or U option follows.
T	the current total result of the $DEFINE command is tested. The test performed depends upon whether the D or U option follows.
D	tests to see if the result specified by the L or T parameters is defined.
U	tests to see if the result specified by the L or T parameters is undefined.
$name	is a name defined via the $LABEL command. If a skip is specified, the skip will be done to this label.

Example:

```
BUILD TEST.CSS
$DEFINE 5,, ST (@1) EXT (=) LU2 CL(L) GOO ST(.FTN)
$WR @*5
$EX
ENDB
```

This $DEFINE command perfoms a check to see if the first positional parameter in the CSS call contains a filename extension. If it does, the following two operations are performed to clear the result of the EXT operator and the $DEFINE is exited.

If no filename extension is specified the following two operators are skipped and an extension is attached.

6.6.6.3 Computation and Conversion Operators

The computation and conversion operators are used to perform decimal or hexadecimal computation and decimal to hexadecimal (or vice-versa) conversion, and then assign the result as the value of the variable specified in the $DEFINE command.

6.6.6.3.1 DCOMPUTE Operator

The DCOMPUTE operator of the $DEFINE command is used to perform decimal computation within a CSS line. The computed value then becomes the value of the variable defined in the $DEFINE command.

Format:

$$\text{\underline{DC}OMPUTE} \left[\begin{Bmatrix}\text{\#digits}\\ \boxed{4}\end{Bmatrix},\right] \text{operand}_0 \left[\left[\text{operator}_1\ \text{operand}_1\right] \left[\text{operator}_n\ \text{operand}_n\right]\right]$$

Parameters:

#digits	specifies the number of digits for the decimal result with leading zeros and including the sign column (+ or -). If not specified the default number of digits used (including sign) is 4.
operand	is the operand (in decimal) with optional sign (+ or -). The range is absolute up to Y'OFFFFFFF'.

operator	is the computational operator:

+	(addition)
-	(subtraction)
*	(multiplication)
/	(division)

Functional Details:

The maximum value allowed for an operand or a result is absolute Y'OFFFFFFF'. Values outside this range generate the following message.

```
    DEF6-ERR
```

Mathematical computation is performed from left to right, and the intermediate result is combined with the next operator and the following operand. Computation is performed according to the fixed point integer rules of rounding.

Examples:

```
    $DEFINE 7,,DCOMPUTE (-33)
```

-033 becomes the value of variable 7(@*7). The default number of digits (4) is used.

```
    $DEFINE 4,,DCOMPUTE (6,-2+5/-2*4)
```

-00004 becomes the value of variable 4 (@*4). The number of digits in the result is defined as 6.

```
    $DEFINE 5,,DC(@*4*@*7+100)
```

+232 becomes the value of variable 5(@*5). This is determined by multiplying the value of variable 4 (@*4), which is defined above, as -4 with the value of variable 7(@*7), which is defined as -33 above then adding 100 to the result. The default number of digits (4) is used.

6.6.6.3.2 DHCONVERT Operator

The DHCONVERT operator of the $DEFINE command is used to perform decimal computation and then convert the result to hexadecimal. This hexadecimal result is then assigned as the value of the variable specified in the $DEFINE command.

Format:

$$\text{DHCONVERT}\left(\left[\begin{Bmatrix}\#\text{digits}\\ 4\end{Bmatrix}\right], \text{operand}_0 \left[\left[\text{operator}_1\ \text{operand}_1\right]\left[\text{operator}_n\text{operand}_n\right]\right]\right)$$

Parameters:

#digits	specifies the number of digits for the hexadecimal result with leading zeros and excluding the sign designator. If not specified the default number of digits is 4.
operand	is the operand (in decimal). Negative numbers are not allowed. The range is absolute up to Y'OFFFFFFF'.
operator	is the computational operator:

+	(addition)
-	(subtraction)
*	(multiplication)
/	(division)

Functional Details:

The maximum value allowed for an operand or a result is absolute Y'OFFFFFFF'. Values outside this maximum generate the following message:

```
DEF7-ERR
```

Mathematical computation is performed from left to right and the immediate result is combined with the next operator and the following operand. Computation is performed according to the fixed point integer rules of rounding.

Examples:

`$DEFINE 7,,DHCONVERT(-33)`	the value of variable 7(@*7) becomes hexadecimal 0021.
`$DEFINE 4,,DHCONVERT(6,-2+5/-2*-4)`	the value of variable 4(@*4) becomes a hexadecimal 000004.
`$DEFINE 5,,DHCONVERT(4*@*7+100)`	the value of variable 5 becomes a hexadecimal 00B8. (4x21)+100 = 184 = 00B8 in hex.

6.6.6.3.3 HCOMPUTE Operator

The HCOMPUTE operator of the $DEFINE command enables a user to perform hexadecimal computation within the $DEFINE command and return the result as the defined variables value.

Format:

$$\text{HCOMPUTE}\left(\left[\begin{Bmatrix}\text{\#digits}\\ \boxed{4}\end{Bmatrix},\right]\ \text{operand}_0\ \left[\left[\text{operator}_1\ \text{operand}_1\right]\ \left[\text{operator}_n\,\text{operand}_n\right]\right]\right)$$

Parameters:

#digits	defines the number of digits for the hexadecimal result with leading zeros. If not specified the default number of digits is 4.
operand	is an operand in hexadecimal without sign (all values are assumed positive), the maximum value being absolute up to Y'OFFFFFFF'.
operator	is one of the following mathematical operators:

+	(addition)
-	(subtraction)
*	(multiplication)
/	(division)

Functional Details:

The range allowed for an operand or a result is up to absolute Y'OFFFFFFF' otherwise the following message is generated:

```
    DEF6-ERR
```

If the hexadecimal result is negative, the following message is generated:

```
    DEF7-ERR
```

Computation within an HCOMPUTE operator is from left to right; the intermediate result is combined with the next operator and the following operand.

Examples:

```
    $DEFINE 7,,HCOMPUTE(AEO)            @*7 = 0AE0

    $DEFINE 4,,HCOMPUTE(6,CO/20+18)     @*4 = 00001E
```

6.6.6.3.4 HDCONVERT Operator

The HDCONVERT operator of the $DEFINE command enables the user to perform hexadecimal computation within a $DEFINE command. The result is converted to decimal and is returned as the value of the defined variable.

Format:

$$\text{HDCONVERT}\left(\left[\begin{Bmatrix}\text{\#digits}\\ 4\end{Bmatrix},\right]\ \text{operand}_0\ \left[\left[\text{operator}_1\ \text{operand}_1\right]\ \left[\text{operator}_n \text{operand}_n\right]\right]\right)$$

Parameters:

#digits	specifies the number of digits for the decimal result with leading zeros and the sign (+ or -). If not specified the default number of digits is 4.

operand	is a hexadecimal operand without sign. The maximum value allowed is absolute Y'0FFFFFFF'.
operator	is one of the following mathematical operators:

+	(addition)
-	(subtraction)
*	(multiplication)
/	(division)

Functional Details:

The maximum allowable value for an operand or a result is absolute Y'0FFFFFFF'. Values greater than the maximum will generate the following message:

```
    DEF6-ERR
```

A negative hexadecimal operand will generate the following message:

```
    DEF7-ERR
```

Computation within the HDCONVERT operator is from left to right, and the intermediate result is always combined with the next operator and the following operand.

Examples:

```
    $DEFINE 7,,HDCONVERT(A0)              @*7 = +160

    $DEFINE 4,,HDCONVERT(6,C0/20+18)      @*4 = +00030
```

6.6.6.4 Other Operators

The following sections detail various miscellaneous operators for the $DEFINE command.

6.6.6.4.1 CLEAR Operator

The CLEAR operator of the $DEFINE command enables the user to clear the current total result or the last result determined in the $DEFINE command.

Format:

$$\underline{\text{C}}\text{LEAR}\left(\begin{Bmatrix} \text{L} \\ \text{T} \end{Bmatrix}\right)$$

Parameters:

L	specifies that the last result determined is to be reset.
T	the current total result is to be reset.

Functional Details:

Use of the CLEAR (L) form of this operator resets the last result, even if a skip was performed. The last result depends on the value the last operator (except logical operators) determined.

Example:

The following is an example of how to add the default extension .FTN to a file descriptor. The file descriptor passed in the CSS call is allowed with or without an extension.

```
    BUILD TEST.CSS
    $DEFINE 5,,ST(@1) EXT (=) LU2 <L(L) GO 0 ST(.FTN)
    $WR @*5
    $EX
    ENDB
```

This example CSS tests to see if an extension is included in the CSS call. If an extension is specified it is not changed. If no extension is specified, the default extension .FTN is added. If this CSS was called with the following:

```
    TEST SYS:ABC              the result @*5 = SYS:ABC.FTN
    TEST BBBB.XYZ             the result @*5 = BBBB.XYZ
```

6.6.6.4.2 CURRENT Operator

The CURRENT operator of the $DEFINE command is used to determine current information within the user's environment and to assign that information as the value of the variable being defined.

Format:

```
           ( BATCH       )
           ( DATE        )
           ( EOT         )
CURRENT  ( ( GROUP       ) )
           ( INTERACTIVE )
           ( PRIVATE     )
           ( TIME        )
           ( USERNAME    )
```

Parameters:

BATCH	in batch mode the value returned is the batch job file descriptor; in interactive mode the value is undefined.
DATE	the value returned is the current date in the format MM/DD/YY or DD/MM/YY depending on the format selected at OS/32 system generation.
EOT	the value returned is the last end of task code generated. A maximum of four digits is allowed. Leading zeros are dropped.
GROUP	the value returned is the 5 digit current group account number with leading zeros.
INTERACTIVE	in interactive mode the value returned is the interactive device name, in batch mode the value is undefined.
PRIVATE	the value returned is the 5 digit current private account number with leading zeros.
TIME	causes the current time (HH:MM:SS) to be returned.
USERNAME	causes the current username to be returned.

Example:

```
    BUILD TIME.CSS
    $DEFINE 5,,CURRENT(TIME)
    $WR @*5
    $EX
    ENDB
```

Execution of this CSS will cause the current time to be written as @*5.

6.6.6.4.3 DVOLUMENAME Operator

The DVOLUMENAME operator of the $DEFINE command enables the user to determine default volume names such as SYSTEM volume, SPOOL volume, etc. and assign the name as the value of the defined variable.

Format:

$$\text{DVOLUMENAME}\left(\begin{Bmatrix}\underline{\text{PR}}\text{IVATE}\\ \underline{\text{R}}\text{OLL}\\ \underline{\text{SP}}\text{OOL}\\ \underline{\text{SY}}\text{STEM}\\ \underline{\text{T}}\text{EMP}\end{Bmatrix}\right)$$

Parameters:

PRIVATE	returns the volume name of the users default volume.
ROLL	returns the volume name of the ROLL volume.
SPOOL	returns the volume name of the SPOOL volume.
SYSTEM	returns the volume name of the SYSTEM volume.
TEMP	returns the volume name of the TEMP volume.

Functional Details:

The volume name returned is always followed by a colon (:).

Examples:

Assume that volume SCRT/TEMP has been set at the system console.

`$DEFINE 6,TEMPVOL,DVOLUMENAME(TEMP)`	
`$WR @*6`	reference by variable would return SCRT:
`$WR @*/TEMPVOL/`	reference by variable name would also return SCRT:

6.6.6.4.4 REQUIRED Operator

The REQUIRED operator of the $DEFINE command enables a user to designate a new internal variable as required; that is, the variable must have a defined value. If the new internal variable designated as REQUIRED is not defined within the CSS, execution of the CSS is paused and the user is prompted at the user's MTM console to supply a definition for the required variable.

Format:

```
    REQUIRED [([name])]
```

Parameters:

name	is an optional 1- to 8-character name for the required new internal variable that MTM will use when the user is prompted at the user's MTM terminal. This name may be composed of any of the letters A through Z.

Functional Details:

The REQUIRED operator must be the last operator in a $DEFINE command. All blanks between the parentheses and between the name are dropped.

The name for the required new internal variable that is displayed to the user console is one of the following (in order of precedence):

- The name specified in the name field of the REQUIRED operator,
- The name used in the $DEFINE command, or
- The number specified in the $DEFINE command.

Examples:

```
    BUILD TEST.CSS
    $DEFINE 3,LISTDEV,REQUIRED
    $DEFINE 4,OPTION,REQUIRED (NEWNAME)
    $DEFINE 5,,REQUIRED
    $EXIT
    ENDB
```

The above CSS identifies three new internal variables (3, 4, and 5) as required variables. If this CSS is called as follows, the following message prompts will be issued at the users console:

*TEST	CSS call without parameters
-GIVE LISTDEV=	Prompt for the first required variable, the variable name is used in the name field
-GIVE NEWNAME=	Prompt for second required variable, the name in REQUIRED field is used
-GIVE IVAR 005=	Prompt for third required variable, the variable number is used

6.6.6.4.5 SEARCH Operator

The SEARCH operator of the $DEFINE command enables the user to perform string searches for matches with specified keywords passed in the CSS call. On each match found, the string (including the keyword) is moved to the value of the new variable defined in the $DEFINE command.

Format:

$$\text{\underline{S}EARCH delimiter}_1 \begin{Bmatrix} \text{'}d_2\text{'} \\ \text{'}d_2+ \end{Bmatrix},\left[\text{keyword}_1 \left[\text{'keyword}_2\text{'...'keyword}_n\right]\right],\left[\text{string}_1 \left[d_2\text{string}_2\right]\right]\text{delimiter}_1$$

Parameters:

delimiter$_1$	is one of the following character pairs used to delimit the SEARCH operator specifications: delimiter$_1$ ...delimiter$_1$ = # ... # ' ... ' + ... + : ... : (...) The character pair chosen as the specification delimiter must not appear in the SEARCH operator specifications or as a string delimiter (d_2).
d_2	is the string delimiter which is used to separate the strings to be searched. The string delimiter may be any character except carriage return or semicolon. If the 'd_2' option is used, the delimiter (d_2) following the matched string is not included when the string is moved, if the 'd_2+ delimiter is used, the delimiter (d_2) is included when the string is moved.
keyword$_1$ [...keyword$_n$]	is a 1- to 8- character (A through Z) keyword. A keyword specification can be further defined to show the minimum number of characters that can be used to reference the keyword. This is accomplished by separating the required characters of the keyword and the optional characters of the keyword with a period. For example: OP.TION The keyword name is OPTION but a call specifying OP= will reference this keyword. Multiple keywords may be defined in a SEARCH operator, all strings are searched for matches with each defined keyword. Multiple keywords are separated by a ' mark.
string$_1$ [...string$_n$]	is a character string which may contain any character except carriage return or semicolon. Null strings are allowed. The specified string is searched for any matches with keywords. If a positional parameter reference is specified (@1, @2) the string to be searched can be passed in the CSS call.

Functional Details:

The beginning of a string is tested for a match with the specified keywords. The search for a match begins with the first string. If one of the defined keywords matches a string entry, this string is moved to the new variable's value. The move includes leading blanks, the keyword, and all following characters up to the next string delimiter (d_2) or including the string delimiter if the 'd_2+ delimiter was specified. This process is repeated for each string to be searched. For example:

If the keyword is:

```
    OPTION
```

and the string delimiter (d_2) is:

```
    '#'
```

and the string to be searched is:

```
    ...# OPT = HOLL BATCH # ...
```

the new variable being defined has a value of:

```
    OPT = HOLL BATCH
```

Examples:

```
    BUILD TEST.CSS
    $DEF 5,,SEARCH('#',OP.TION'BA.TCH, @1)
    $WR @*5
    $EX
    ENDB
```

The above CSS identifies the pound sign as the string delimiter; keywords are OP.TION and BA.TCH; the string to be searched is @1, the first parameter passed in the CSS call.

When calling the above CSS with the following call:

```
    TEST OP/AAAA# BATCH # SOURCE
```

The first string searched is OP/AAAA. A match with the first keyword is found OP.TION. OP/AAAA is moved to the variables value.

The next string searched is BATCH. A match with the second keyword is found BA.TCH. BATCH is moved to the variables value. next string searched is SOURCE. No match is found.

The subsequent value of $WR @*5 is OP/AAAA BATCH

If calling TEST.CSS with the following:

```
    TEST xx # BATCH # BA/AAA # YY # OPTI
```

The first string searched (xx) has no match. The second string searched (BATCH) matches a keyword. The third string seached (BA/AAA) matches a keyword. The fourth string (YY) has no match. The fifth string searched (OPTI) matches a keyword.

The subsequent value of $WR @*5 = BATCH BA/AAA OPTI

6.6.6.4.6 STRING Operator

The STRING operator of the $DEFINE command enables the value of the new variable being defined to be a user specified string.

Format:

STRING delimiter$_1$ string delimiter$_1$ [...delimiter$_n$ string delimiter$_n$]

Parameters:

delimiter	is any of the following characters that delimits the beginning and end of the string:

```
#...#
'...'
+...+
:...:
(...)
```

The character used as the delimiter should never appear within the string.

string	is a character string which may contain any characters except carriage return or the delimiter character. This string becomes the value of the new variable being defined in the $DEFINE command. Leading and/or trailing blanks are included.

Example:

```
    BUILD TEST
    $DEFINE 7,, STRING (ABC) ST # A ($$) A#
    $WR [@*7]
    $EX
    ENDB
```

Calling the above CSS with the following call:

```
    *TEST
```

The resulting output of the $WR @*7 statement is:

```
    [ABC A ($$) A]
```

6.6.7 $EXIT Command

The $EXIT command terminates a CSS procedure. Control is returned to the calling CSS procedure or the terminal if the CSS procedure was called from the terminal. All commands on the lines after the $EXIT command are ignored.

Format:

```
    $EXIT
```

6.6.8 $FREE Command

The $FREE command frees one or more local or global variables. This command has no effect on new global or new internal variables.

Format:

```
$FREE varname₁ [,...,varnameₙ]
```

Parameters:

varname	is a 1- to 8-character name specifying the variable whose name and value are to be freed.

Example:

```
$FREE @A
```

6.6.9 $GLOBAL Command

The $GLOBAL command names a global variable and specifies the maximum length of the variable to which it can be set by the $SET command.

Format:

$$\text{\$GLOBAL varname} \left[\left(\begin{Bmatrix} \text{length} \\ 8 \end{Bmatrix} \right) \right] \left[,\ldots,\text{varname} \left[\left(\begin{Bmatrix} \text{length} \\ 8 \end{Bmatrix} \right) \right] \right]$$

Parameters:

varname	is a 1- to 8-character name (the first character is alphabetic) preceded by the @ sign, identifying a global variable.
length	is a decimal number from 4 through 32 specifying the length of the variable defined by the $SET command. If this parameter is omitted, the default is 8.

Example:

```
$GLOBAL @A(6)
```

6.6.10 $JOB and $TERMJOB Commands

The $JOB and $TERMJOB commands set the boundaries of a CSS job. The $JOB command indicates the start, and the $TERMJOB command indicates the end of a CSS job that contains all the user CSS commands.

Format:

```
$JOB
     CPUTIME=maxtime
    [,classid=iocount1]  [,...,classid=iocount32]
   .
   .
   .
$TERMJOB
```

Parameters:

CPUTIME=	maxtime is a decimal number specifying the maximum CPU time to which the CSS routine is limited. If this parameter is omitted, the default established at MTM sysgen is used. If 0 is specified, no limits are applied.
classid=	is one of the 4-character alphanumeric mnemonics specified at MTM sysgen that is associated with each specified device or file class.
iocount	is a decimal number specifying the maximum number of I/O transfers to which the CSS routine is limited for that class. If this parameter is omitted, the default established at sysgen time is used. If 0 is specified, no limits are applied to that class.

Functional Details:

The $JOB and $TERMJOB commands are not necessary in a CSS procedure. However, they help prevent errors in one CSS job from affecting other CSS jobs. If a CSS job contains an error, the statements remaining in that job are skipped until a $TERMJOB command is found. The next command executed is the first command found after a $TERMJOB command. If the next command is a $JOB command signifying the start of a new CSS job, it could be skipped because the system is looking for a $TERMJOB that signifies the end of the CSS job containing the error.

The CSS job containing an error is aborted, and the end of task code is 255. The $JOB command resets the end of task code to 0 for the next CSS job.

Interactive jobs have no default limits established at sysgen time. However, the user can specify CPU time and I/O transfer limits for a particular job through the $JOB command.

Any limits in the $JOB command found in a batch stream are ignored if limits were already specified in the SIGNON command.

6.6.11 $LOCAL Command

The $LOCAL command names a local variable and specifies the maximum length variable to which it can be set by the $SET command.

Format:

$$\text{\$LOCAL varname}\left[\left(\begin{Bmatrix}\text{length}\\ \mathbf{8}\end{Bmatrix}\right)\right]\left[\text{,...,varname}\left[\left(\begin{Bmatrix}\text{length}\\ \mathbf{8}\end{Bmatrix}\right)\right]\right]$$

Parameters:

varname	is a 1- to 8-character name (the first character is alphabetic) preceded by the @ sign, identifying a local variable.
length	is a decimal number from 4 through 32 specifying the length of the variable defined by the $SET command. If this parameter is omitted, the default is 8.

Example:

```
$LOCAL @A(4)
```

6.6.12 $PAUSE Command

The $PAUSE command suspends execution of a CSS procedure.

Format:

```
    $PAUSE
```

Functional Details:

When $PAUSE is entered, the CSS procedure remains suspended until the $CONTINUE command is entered or the $CLEAR command is entered to terminate a procedure suspended by a $PAUSE.

6.6.13 PRIOR Command

The PRIOR command is used in CSS files to set the priority for a subsequently loaded task. This command is available in CSS files from the system account and from privileged users of MTM (to raise or lower the priority of a susbsequently loaded task) and to nonprivileged MTM users (to lower the priority of a subsequently loaded task relative to the user's MTM priority.) However, nonprivileged users of MTM cannot use the PRIOR command to raise the priority of a task above their MTM priority.

Format:

```
    PRIOR n
```

Parameter:

n	is a decimal number specifying the priority of the susbsequently loaded task relative to the priority of MTM. n may range from 1 through 255 when the PRIOR command is in a CSS file from the system account or from a privileged user. n may range from 12 through 255 when the PRIOR command is in a CSS file from a nonprivileged MTM user.

Functional Details:

The PRIOR command can be entered from CSS files only. If the task loaded subsequent to a PRIOR command generates a load error or goes to end of task, the priority specified in the PRIOR command is reset to the default MTM priority.

If an invalid priority number is specified in a PRIOR command (i.e. 1-11 by a nonprivileged user), the invalid priority specification is ignored, no message is generated, and the default MTM priority is used.

6.6.14 $RELEASE Command

The $RELEASE command is used to release a new global or new internal variable from its current value and delete the released variable's associated buffer. This command has no effect on local or global variables.

Format:

$$\underline{\$REL}\text{EASE} \begin{Bmatrix} \underline{\text{G}}\text{VARIABLE} \\ \underline{\text{I}}\text{VARIABLE} \end{Bmatrix} \left[, \begin{Bmatrix} n_1/n_2 \\ n_1 \; , \ldots , n_n \\ \text{ALL} \end{Bmatrix} \right]$$

Parameters:

GVARIABLE	indicates that the variables to be released are new global variables.
IVARIABLE	indicates that the variables to be released are new internal variables.
n_1/n_2	indicates that all variables (of the type selected via the preceding parameter) between the range n_1/n_n be released. Where n is a decimal number between 1 and the maximum value allowed at MTM sysgen for the specified variable type.
$n_1 \ldots n_n$	n is a decimal number of a variable (either new global or new internal) or variables to be released. n must be within the range 1 and the maximum value allowed at MTM sysgen for the specified variable type.
ALL	specifies that all new internal or new global variables be released. This is the default if no specific variable numbers are specified.

Functional Details:

This command may be entered in command mode, task loaded mode, task executing mode, and CSS mode. In order to reduce buffer overhead, variables that are no longer being used should be released. If this command is directed to a variable that was already released, the command is ignored and no error message is generated.

Examples:

```
    $RELEASE GVARIABLE, 1/5
```

All new global variables from 1 through 5 are released.

```
    $RELEASE IVARIABLE, 16, 19, 18, 25
```

The new internal variables numbered 16, 19, 18, and 25 are released.

```
    $RELEASE IVARIABLE, ALL
```

All new internal variables are released.

NOTE

This command does not release local and global variables created with the $SET command.

6.6.15 $SET Command

The $SET command establishes the value of a named local or global variable. This command has no effect on new global or new internal variables.

Format:

```
$SET varname=e
```

Parameter:

varname=	e is an expression, variable, or parameter established as the value of the variable.

Functional Details:

Expressions for this command are concatenations of variables, parameters, and character strings. No operators are allowed in an expression. If a character string is included in an expression, it must be enclosed between apostrophes ('). If an apostrophe is part of the character string, it must be represented as two apostrophes ('').

The initial value of the variable is blanks. This allows the $IFNULL and $IFNNULL commands to test for a null or not null value.

Examples:

```
$SET @A = @A1@A2

$SET @A = @1

$SET @A = 'A''B'
```

6.6.16 SET CODE Command

The SET CODE command modifies the current end of task code.

Format:

```
    SET CODE n
```

Parameter:

n	is a decimal number from 1 through 254.

6.6.17 $SKIP Command

The $SKIP command is used between the $JOB and $TERMJOB commands. The $SKIP command indicates that subsequent commands are to be skipped until a $TERMJOB command is found. The end of task code is set to 255.

Format:

```
    $SKIP
```

6.6.18 $WAIT Command

The $WAIT command suspends execution of a CSS for a specified period of time.

The $CONTINUE command can be used to override this command and continue the CSS.

Format:

$$\text{\$WAIT}\left[\begin{Bmatrix} n \\ 1 \end{Bmatrix}\right]$$

Parameter:

n	is a decimal number from 1 through 900 specifying the number of seconds CSS execution will be suspended. If this parameter is omitted, the default is 1 second.

Functional Details:

The $WAIT command will only function from a CSS routine.

The $CONTINUE command can be used to override this command and continue the CSS.

6.6.19 $WRITE Command

The $WRITE command writes a message to the terminal or log device for both interactive and batch jobs.

Format:

```
    $WRITE text [;]
```

Functional Details:

The message is output to the terminal or log device. It begins with the first nonblank character after $WRITE and ends with a semicolon or carriage return. The semicolon is not printed.

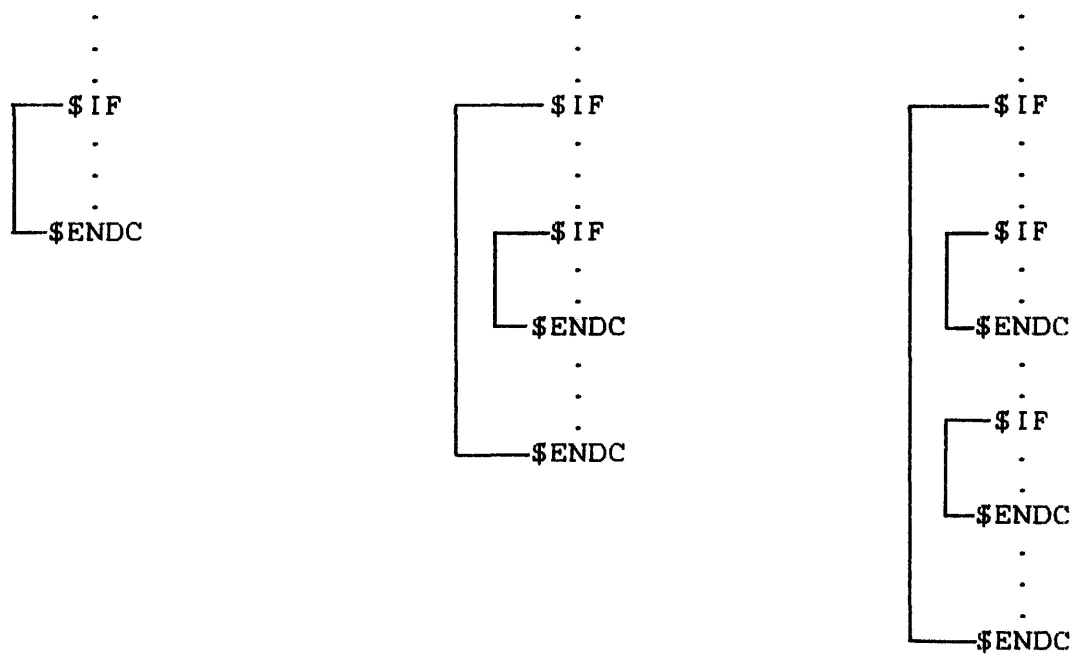
6.7 LOGICAL IF COMMANDS

The logical IF commands all start with the three characters, $IF, and allow one argument; e.g., $IFE 225, $IFX B.CSS, $IFNULL @1.

Each logical IF command establishes a condition that is tested by the CSS processor. If the result of this test is true, commands up to a corresponding $ELSE or $ENDC command are executed. If the result is false, these same commands are skipped.

The $ENDC command delimits the range of a logical IF; however, nesting is permitted so each $IF must have a corresponding $ENDC.

In the following examples, the ranges of the various logical IF commands are indicated by brackets:



There is no restriction on the depth of nesting. Logical IF commands are used within a CSS file. However, they differ from previous CSS commands in that each one tests a specific built-in, defined condition rather than causes a specific action.

The logical IF commands fall into three categories:

- End of task code testing
- File existence testing
- Parameter existence testing

6.7.1 End of Task Code Testing Commands

The end of task code is a halfword quantity maintained for each user by the system. It is set or reset in any of the following ways:

SET CODE n	This command, which can be included in a CSS file or entered at the terminal, sets the end of task code to n.
$JOB	As part of its start job function, this command resets the end of task code for the current CSS task to 0.
Command error	A command error causes the CSS mechanism to skip to $TERMJOB assuming that a $JOB was executed. (If no $JOB was executed, CSS terminates.) To indicate that the skip took place, the end of task code is set to 255.
$SKIP	This command has the same effect as a command error.
End of task (SVC 3,n)	When any task terminates by executing the end of task program command (SVC 3,n), the end of task code for that task is set to n.
CANCEL	When a task is cancelled, the end of task code is set to 255.

The six commands available for testing the current end of task code are as follows:

$IFE n	Test if end of task code is equal to n.
$IFNE n	Test if end of task code is not equal to n.
$IFL n	Test if end of task code is less than n.
$IFNL n	Test if end of task code is not less than n.
$IFG n	Test if end of task code is greater than n.
$IFNG n	Test if end of task code is not greater than n.

In all cases, if the results of the test are "false", CSS skips commands until the corresponding $ELSE or $ENDC. If a CSS attempts to skip beyond EOF, a command error is generated.

6.7.2 File Existence Testing Commands

There are two commands dealing with file existence:

$IFX fd	Test fd for existence
$IFNX fd	Test fd for nonexistence

If the result of the test is false, CSS skips to the corresponding $ELSE or $ENDC command. If a CSS attempts to skip beyond EOF, an error is generated.

If the file descriptor is omitted when entering $IFX, the result is always considered false. If $IFNX is entered without the fd, the result is always considered true.

6.7.3 Parameter Existence Testing Commands

There are two commands dealing with the existence of parameters:

$IFNULL @n	Test if @n is null
$IFNNULL @n	Test if @n is not null

If the result of the test is false, CSS skips to the corresponding $ELSE or $ENDC command. If such skipping attempts to skip beyond EOF, a command error is given.

The use of the multiple @ notation to test for the existence of higher level parameters is permitted. In addition, a combination of parameters can be tested simultaneously.

Example:

```
$IFNU @1@2@3
```

In effect, this tests the logical AND of @1, @2, and @3 for nullity. If any of the three is present, the test result is false.

6.7.4 $ELSE Command

The $ELSE command is used between the $IF and $ENDC command to test the opposite condition of that tested by $IF. Thus, if the condition tested by $IF is true, $ELSE causes commands to be skipped up to the corresponding $ENDC. If the condition is false, $ELSE terminates skipping and causes command execution to resume.

Format:

```
    $ELSE
```

6.8 $GOTO AND $LABEL COMMANDS

The $GOTO command is used to skip forward within a CSS procedure. The $LABEL is used to define the object of a $GOTO.

Format:

```
$GOTO  label

$LABEL label
```

Parameters:

label	is from 1- to 8-alphanumeric characters, the first of which must be alphabetic.

Functional Details:

The $GOTO command causes all subsequent commands to be ignored until a $LABEL command with the same label as the $GOTO command is encountered. At that point, command execution resumes.

The $GOTO cannot branch into a logical IF command range but can branch out from one.

An example of an illegal $GOTO is:

```
$IF      Condition
$GOTO    OUTIF
   .
   .
   .
$ENDC
$IF      Condition
$LABEL   OUTIF
```

The $LABEL occurs within an IF block (the second IF condition) that was not active when $GOTO was executed.

The following is valid, however:

```
    $IF       Condition
    $GOTO     OUTIF
       .
       .
       .
    $ENDC
    $IF       Condition
       .
       .
       .
    $ENDC
    $LABEL     OUTIF
```

6.9 $IFEXTENSION COMMAND

The $IFEXTENSION command is used to test for the existence of an extension for a given fd. If the extension exists, subsequent commands are executed up to the next $ELSE or $ENDC command. If an extension does not exist, subsequent commands are skipped up to the next $ELSE or $ENDC command.

Format:

```
    $IFEXTENSION fd
```

Parameter:

fd	is the file descriptor to be tested to determine if an extension is included.

Functional Details:

$IFEX (with no fd) is always considered false. $IFNEX (with no fd) is always considered true.

6.10 $IFVOLUME COMMAND

The $IFVOLUME command tests for the existence of a volume name in an fd. If a volume exists, subsequent commands are executed up to the next $ELSE or $ENDC command. If the volume is omitted in the fd, subsequent commands are skipped up to the next $ELSE or $ENDC command.

Format:

```
    $IFVOLUME fd
```

Parameter:

fd	is the file descriptor tested to determine if a volume name is included.

6.11 LOGICAL IF COMMANDS COMPARING TWO ARGUMENTS

The following logical IF commands are used to compare two arguments. They differ from the other logical IF commands in that they do not test specific built-in conditions but, rather, test conditions provided by the user. These commands are available only with MTM.

```
$IF . . . EQUAL
$IF . . . NEQUAL
$IF . . . GREATER
$IF . . . NGREATER
$IF . . . LESS
$IF . . . NLESS
```

For each of the logical commands, two arguments are compared according to the mode. There are three valid modes:

- Character
- Decimal
- Hexadecimal

For character mode, the comparison is left-to-right and is terminated on the first pair of characters that are not the same. If one string is exhausted before the other, the short string is less than the long string. If both strings are exhausted at the same time, they are equal. For character mode, the arguments can be enclosed in double quotes if they contain blanks. The quotes are not included in the compare.

For decimal and hexadecimal mode, the comparison is performed by comparing the binary value of the numbers.

If after comparing the arguments for each of the commands, the condition is determined to be true, subsequent commands are executed up to the corresponding $ELSE and $ENDC. If the condition is false, commands are skipped up to the corresponding $ELSE or $ENDC.

6.11.1 $IF...EQUAL, $IF...NEQUAL Commands

The $IF...EQUAL command is used to determine if two arguments are equal, while the $IF...NEQUAL is used to determine if two arguments are not equal.

Format:

$$\text{\$IF} \left\{ \begin{array}{c} \underline{\text{CH}}\text{ARACTER} \\ \underline{\text{D}}\text{ECIMAL} \\ \underline{\text{H}}\text{EXADECIMAL} \end{array} \right\} \text{arg}_1 \ \underline{\text{EQ}}\text{UAL} \ \text{arg}_2$$

$$\text{\$IF} \left\{ \begin{array}{c} \underline{\text{CH}}\text{ARACTER} \\ \underline{\text{D}}\text{ECIMAL} \\ \underline{\text{H}}\text{EXADECIMAL} \end{array} \right\} \text{arg}_1 \ \underline{\text{NE}}\text{QUAL} \ \text{arg}_2$$

6.11.2 $IF...GREATER, $IF...NGREATER Commands

The $IF...GREATER command is used to determine if arg_1 is greater than arg_2. The $IF...NGREATER command is used to determine if arg_1 is not greater than arg_2.

Format:

$$\text{\$IF} \left\{ \begin{array}{c} \underline{\text{CH}}\text{ARACTER} \\ \underline{\text{D}}\text{ECIMAL} \\ \underline{\text{H}}\text{EXADECIMAL} \end{array} \right\} \text{arg}_1 \ \underline{\text{GR}}\text{EATER} \ \text{arg}_2$$

$$\text{\$IF} \left\{ \begin{array}{c} \underline{\text{CH}}\text{ARACTER} \\ \underline{\text{D}}\text{ECIMAL} \\ \underline{\text{H}}\text{EXADECIMAL} \end{array} \right\} \text{arg}_1 \ \underline{\text{NG}}\text{REATER} \ \text{arg}_2$$

6.11.3 $IF...LESS, $IF...NLESS Commands

The $IF...LESS command is used to determine if arg_1 is less than arg_2. The $IF...NLESS command is used to determine if arg_1 is not less than arg_2.

Format:

$$\text{\$IF} \begin{Bmatrix} \underline{\text{C}}\text{HARACTER} \\ \underline{\text{D}}\text{ECIMAL} \\ \underline{\text{H}}\text{EXADECIMAL} \end{Bmatrix} \text{arg}_1 \ \underline{\text{L}}\text{ESS arg}_2$$

$$\text{\$IF} \begin{Bmatrix} \underline{\text{C}}\text{HARACTER} \\ \underline{\text{D}}\text{ECIMAL} \\ \underline{\text{H}}\text{EXADECIMAL} \end{Bmatrix} \text{arg}_1 \ \underline{\text{N}}\text{LESS arg}_2$$

CHAPTER 7
SPOOLING

7.1 INTRODUCTION

The OS/32 Package (Revision 6.2 or higher) now comes with two spooler tasks:

- the OS/32 spooler, and
- the SPL/32 spooler

Both spoolers offer input and output spooling capabilites to the MTM user. The SPL/32 spooler offers a more extensive range of features and capabilities than the OS/32 spooler. The system administrator determines which spooler will be used on a system by selecting the appropriate sysgen statement. Only one spooler can be active on the system at any given time. The OS/32 System Generation (SYSGEN) Reference Manual presents detailed information regarding the procedures for sysgening either spooler.

NOTE

The manner in which pseudo devices are specified and used in the spooling environment differs among the two spoolers. Pseudo devices created for the OS/32 spooler are not compatible with pseudo devices created for the SPL/32 spooler. Do not attempt to mix the various pseudo device types.

7.2 THE OS/32 SPOOLER

The OS/32 spooler is Perkin-Elmer's first generation spooler and until this release was the only spooler available with OS/32. This spooler provides basic input and output spooling services with minimal flexibility and control over the spooling environment. The following sections detail the manner in which an MTM user can utilize the spooling capabilities of OS/32 spooling.

7.2.1 Input Spooling

Input spooling is a process whereby a card deck of information (such as source programs, operator commands, command substitution system (CSS) files, or user data, is copied into a disk file for immediate or subsequent processing.

7.2.2 Input Spooling Control Card Statements

Each batch of cards to be spooled to disk must be preceded by a control card statement. This statement specifies the fd to which the input data (card file) is to be spooled. The OS/32 spooler provides two such control statements:

- /@INPUT
- /@SUBMIT

7.2.2.1 The /@INPUT Control Statement

The /@INPUT control statement is used to copy a card file to a specified fd on disk. The resulting file can be explicitly assigned and read by the user in order to access the spooled information.

Format:

```
    /@INPUT  fd/actno [,DELETE]
```

Parameters:

fd	is the file descriptor of the disk file in the form of voln:filename.ext. The only required field is filename. If voln is omitted, the default spool volume is used.
actno	is the account number the terminal user signs on with.
DELETE	specifies that if a file with the same name and account number already exists, that file is deleted and reallocated.

CAUTION

IF THE WRONG ACCOUNT NUMBER IS ENTERED, THE USER MIGHT DELETE ANOTHER USER FILE.

Example:

A task requires five input data records in order to execute. In the following example, TEST.DTA in account 12 is identified as the file to which the five data records are to be spooled. If the file TEST.DTA currently exists on disk it will be deleted and reallocated as specified by the DELETE option in the /@INPUT statement.

```
/@IN TEST.DTA/12,DELETE
4 INPUT TEST
122736
545627
889710
632192
/@
```

7.2.2.2 The /@SUBMIT Control Statement

The spooler can also be used to submit batch jobs to MTM. This is done through the /@SUBMIT control statement. This statement copies a card file to disk and then submits the file as a batch job. The commands located within the spooled batch file are executed in sequence. The file remains on the disk after execution.

To add batch jobs to the batch queue via the spooler, submit a control statement card with the following format:

Format:

```
/@SUBMIT  fd/actno [,DELETE]
```

Parameters:

fd	is the name of the command file; i.e., the batch job, that is to be placed on the batch queue.
actno	is the account number the terminal user signs on with.
DELETE	specifies that if a file with the same name and account number exists, that file is to be deleted and reallocated.

The end of a card file is signified by placing the symbols /@ in columns 1 and 2 of the last card in the file.

Refer to the OS/32 System Support Utilities Reference Manual for more detailed information on the OS/32 spooler.

The following examples are presented to illustrate two methods of submitting a batch job through the OS/32 spooler.

Method 1:

First, a CSS file named DATA is copied from a card file to a disk file named TEST.CSS on account number 12 on the default spool volume. If TEST.CSS already exists, it is deleted and reallocated. This is done as follows:

```
/@INPUT TEST.CSS/12,DELETE
LO DATA
AS 1,DATA.DTA
AS 3,PR:
AS 5,MAG1:
START
/@
```

The CSS file TEST.CSS created with the previous /@INPUT statement now can be submitted as a batch job named TEST.JOB via the /@SUBMIT control statement. If a file already exists on the disk with the name TEST.JOB, it is deleted and reallocated. When running concurrent batch jobs, each signon ID must be unique.

```
/@SUBMIT TEST.JOB/12,DELETE
SIGNON ME,12,PASSWD
LOG PR:
TEST.CSS
SIGNOFF
/@
```

Method 2:

The procedures shown in Method 1 can also be performed in one step, as the following example shows. In this example the process of creating a CSS file and then submitting the CSS file as a batch job is combined into one step. If the file TEST.JOB already exists on the disk, it is deleted and reallocated. After this batch job completes, the file TEST.JOB remains on the disk.

```
/@SUBMIT TEST.JOB/12,DELETE
SIGNON ME,12,PASSWD
LOG PR:
LO DATA
AS 1,DATA.DTA
AS 3,PR:
AS 5,MAG1:
START
SIGNOFF
/@
```

7.2.3 Output Spooling

Output spooling is a process in which information destined for a physical output device, such as a printer or card punch, is initially copied to a disk file. This file is then copied by the spooler to the physical output device on a task priority basis. This process enables multiple tasks to be generating output for the same output device since output is not routed directly to the device as it is generated.

To make use of the output OS/32 spooler, assign any logical units (lu) to be printed or punched to one or more pseudo devices. As soon as the lu is closed, the OS/32 spooler automatically will print or punch the results. Printing or punching may be delayed because of a backlog to the device.

There is no limit to the number of tasks or logical units that can be assigned to a pseudo device. After the user makes an lu assignment to a pseudo device, the following occurs internally: the operating system automatically intercepts all assignments to that pseudo device and allocates an indexed file called a spool file on the spool volume. Subsequent output calls cause data to be written to this file and not to the device. The spooler supports both image and formatted writes.

When the lu assigned to the spool file is closed, the filename, task name, and priority are placed into the spooler print or punch queue. The queue is maintained as a file on the spool volume. If there is an entry on the queue, the output spooler begins printing or punching and stays active as long as there is something on the queue. Files are spooled and output on a task priority basis. The user must ensure that sufficient disk space is available to accommodate output spooling. The user task is responsible for handling end of medium (EOM) status while writing to spool files within their own standard I/O error recovery routines.

Printing multiple copies of a disk file or punching multiple copies of a card deck is accomplished through use of the spooler. To print or punch a disk file using the spooler, issue a command through MTM from the terminal. This is done with the PRINT and PUNCH commands. See Sections 2.38 and 2.39.

If the device specified in a PRINT or PUNCH command does not support printed output or output punching respectively, the output will be generated in the way that is supported on the specified device.

For print files, a header page precedes each file printed. The header page has the format:

```
USERID

ACCOUNT NUMBER

TIME OF DAY

DATE
```

When a file is directed to a card punch file, each output record is 80 bytes in length. A header card precedes the punched output; a trailer card terminates the punched output. Header suppression is not supplied.

Example:

To list and punch a file named TEST.CSS in account number 12 on the volume MTM using the OS/32 spooler, enter:

```
SIGNON ME,12,MEPASS
PRINT MTM:TEST.CSS
PUNCH MTM:TEST.CSS
SIGNOFF
```

The header page for the print examples reads:

```
TEST
AC=00012
14:36:50
07/08/77
```

7.2.4 Spooling Errors

The following message is generated by the operating system in response to a spooler command.

```
FILE voln:filename.ext/acct  NOT ENTERED ONTO PRINT QUEUE
```

A spool file was closed but the spooler task was not loaded or started. The system operator can reenter a .SPL PRINT command when the spooler is started.

7.3 THE SPL/32 SPOOLER

The SPL/32 spooler is the latest spooling product offered with the OS/32 operating system. SPL/32 will only execute on systems running Revision R06.2 or higher of OS/32.

SPL/32 offers increased flexibility in creating and controlling the spooling environment of a system. Some of the features of SPL/32 include:

- The number of output devices is dependent only on the amount of available memory.
- Capability of retaining a spooled output file after it is sent to a device.
- Capability of holding spooled files from output processing.
- The option to backspace, forward space, or rewind a file that is currently being output by the spooler, and then resume output.
- The option to produce up to 255 copies of an output file.
- The option to print informative header and trailer pages to identify output files.
- The capability of using preprinted forms and testing for form alignment before output.
- The capability to alter the output requirements of a file waiting to be output.
- The capability to alter the order in which files are output.
- The capability to control devices within the output spooling environment.
- The capability to quiesce the entire output spooling function or individual devices in an orderly fashion.
- The capability to add or drop spool devices dynamically.

7.3.1 SPL/32 and MTM Interaction

The SPL/32 capabilities available to an MTM terminal user are directly dependent upon the manner in which the spooling environment is configured. MTM users of SPL/32 should refer to the SPL/32 Administration and Reference Manual for specific details on the commands and configurational considerations of using SPL/32.

In general, MTM should be designated the primary control task for SPL/32. This will enable all SPL/32 spooling facilities at the MTM terminal level.

APPENDIX A
MTM COMMAND SUMMARY

```
              { CONTIGUOUS,fsize [,{keys }]                                                  }
              {                    {0000}                                                    }
              {                                                                              }
              { EC [/[{bsize}]] [/[{isize}]] [,[{keys}]]                                     }
              {        {64   }      {3    }      {0000}                                      }
              {                                                                              }
ALLOCATE fd,  { INDEX [,[{lrecl}]] [/[{bsize}]] [/[{isize}]] [,[{keys}]]                     }
              {           {126  }      {1    }      {1    }      {0000}                      }
              {                                                                              }
              { NB [,[{lrecl}]] [/[{bsize}]] [/[{isize}]] [,[{keys}]]                        }
              {        {126  }      {64   }      {3    }      {0000}                         }
              {                                                                              }
              { ITAM [,[{lrecl}]] [/[{bsize}]] [,[{keys}]]                                   }
              {          {126  }      {1    }      {0000}                                    }


                     [ [ {access privileges} ]                ] [ { SVC15  } ]
                     [ [ {      SRW        } ]                ] [ { SVCF   } ]
ASSIGN lu,fd         [,[ {      SREW       } ]  [,{keys}]     ] [,{ VFC    } ]
                     [ [ {      SRO        } ]    {0000}      ] [ { HI     } ]
                     [ [ {      ERW        } ]                ] [ { LOW    } ]
                     [                                        ] [ { MEDIUM } ]

BFILE [fd,] lu

BIAS {address}
     {   *   }

BREAK

BRECORD [fd,]  lu
```

```
       (fd)
BUILD  {  }  [,APPEND]
       (lu)
   .
   .
   .
ENDB

CANCEL

       (lu1  [,lu2,...lun])
CLOSE  {                  }
       (ALL               )

CONTINUE [address]

DELETE fd1  [,fd2,...,fdn]

                        [  [(n1/n2)]    [(      fd      )]]
DISPLAY  (GVARIABLE)    [  [(     )]    [(              )]]
         {         }    [, [{  n  }]  , [{              }]]
         (IVARIABLE)    [  [(     )]    [( user console )]]
                        [  [( ALL )]    [(              )]]

                        [ (     fd      )]
DISPLAY ACCOUNTING      [,{             }]
                        [ (user console )]

                        [ (     fd      )]
DISPLAY DEVICES         [,{             }]
                        [ (user console )]

                        [ (     fd      )]
DISPLAY DFLOAT          [,{             }]
                        [ (user console )]
```

```
                          ┌ ⎧          :          ⎫ ┐
DISPLAY FILES ,           │ ⎨        voln:        ⎬ │ [filename] [.[ext]]
                          └ ⎩default user vol     ⎭ ┘

                             ┌ ┌ ⎧ F ⎫ ┐ ┐
                             │ │ ⎪ S ⎪ │ │  ┌  ⎧      fd      ⎫ ┐
                             │/│ ⎨ G ⎬ │ │  │ ,⎨              ⎬ │
                             │ │ ⎪ N ⎪ │ │  └  ⎩ user console ⎭ ┘
                             │ │ ⎪ O ⎪ │ │
                             └ └ ⎩ L ⎭ ┘ ┘

                         ┌  ⎧      fd      ⎫ ┐
DISPLAY FLOAT            │ ,⎨              ⎬ │
                         └  ⎩ user console ⎭ ┘

                      ┌  ⎧      fd      ⎫ ┐
DISPLAY LU            │ ,⎨              ⎬ │
                      └  ⎩ user console ⎭ ┘

                              ┌  ⎧      fd      ⎫ ┐
DISPLAY PARAMETERS            │ ,⎨              ⎬ │
                              └  ⎩ user console ⎭ ┘

                             ┌  ⎧      fd      ⎫ ┐
DISPLAY REGISTERS            │ ,⎨              ⎬ │
                             └  ⎩ user console ⎭ ┘

                        ┌  ⎧      fd      ⎫ ┐
DISPLAY TIME            │ ,⎨              ⎬ │
                        └  ⎩ user console ⎭ ┘

                         ┌  ⎧      fd      ⎫ ┐
DISPLAY USERS            │ ,⎨              ⎬ │
                         └  ⎩ user console ⎭ ┘
```

```
        (MESSAGE  )
        (PROMPT   )
ENABLE  (ETM      )
        ($VARIABLE)

                   [(,n          )]  [ (    fd       )]
EXAMINE address1   [(/address2   )]  [,(             )]
                   [(,1          )]  [ (user console )]

FFILE [fd,] lu

FRECORD [fd,] lu

      [(mnemonic)]
HELP  [(   *    )]

        [ (segsize increment)]
INIT fd [,(                 )]
        [ (        1        )]

          [   (   ,fd1      )]
INQUIRE   [fd (             )]
          [   (user console )]

LOAD [taskid,] fd [,segsize increment]  [,SCTASK]

           [ [(NOCOPY)]]   [(n )]
LOG  [fd]  [,[(      )]] , [(  )]
           [ [(COPY  )]]   [(15)]

               [ [(NOCOPY)]]   [(n )]
SET LOG  [fd]  [,[(      )]] , [(  )]
               [ [(COPY  )]]   [(15)]

         (userid   )
MESSAGE  (         )  message
         (.OPERATOR)
```

```
                      ┌ ┌ data1 ┐ ┐
MODIFY address,       │ {       } │  [,data2,...,datan]
                      └ └   0   ┘ ┘

          ┌ ┌ AFPAUSE     ┐ ┐   ┌  ┌ SVCPAUSE     ┐ ┐
OPTIONS   │ {             } │   │ ,{              } │   [,NONRESIDENT]
          └ └ AFCONTINUE  ┘ ┘   └  └ SVCCONTINUE  ┘ ┘

PASSWORD current password, new password

PAUSE

           ┌ MESSAGE   ┐
           │ PROMPT    │
PREVENT   {            }
           │ ETM       │
           └ $VARIABLE ┘

PRINT fd [,DEVICE=pseudo device] [,COPIES=n] [,DELETE] [,VFC]

PUNCH fd [,DEVICE=pseudo device] [,COPIES=n] [,DELETE] [,VFC]

PURGE fd

                                 ┌  ┌      n1/n2          ┐ ┐
$RELEASE  ┌ GVARIABLE ┐          │  │                     │ │
          {           }          │ ,{  n1 [,...,nn]       } │
          └ IVARIABLE ┘          │  │                     │ │
                                 └  └      ALL            ┘ ┘

RENAME oldfd,newfd

REPROTECT fd,new keys

REWIND [fd,] lu

    or

RW [fd,] lu
```

```
                   ⎧       ⎧actno₁ [/{RW}] [,...,max actno [/{RW}]]⎫⎫
                   ⎪       ⎪          {RO}                     {RO}  ⎪⎪
                   ⎪ ADD,  ⎨                                          ⎬⎪
                   ⎪       ⎪                     ALL [/{RW}]           ⎪⎪
                   ⎪       ⎩                           {RO}            ⎭⎪
                   ⎪                                                     ⎪
RVOLUME voln,      ⎨ REMOVE, { actno₁ ,...,max actno }                   ⎬
                   ⎪         {          ALL          }                   ⎪
                   ⎪                                                     ⎪
                   ⎪           ⎡ ⎧      actno        ⎫⎤                  ⎪
                   ⎪ USERS     ⎢,⎨ actno₁ - actno₂   ⎬⎥                  ⎪
                   ⎩           ⎣ ⎩   0-max actno     ⎭⎦                  ⎭

SEND message [;]

SET GROUP n

SET KEYOPERATOR [character]

SET PRIVATE n

SIGNOFF

SIGNON userid,actno,password [,ENVIRONMENT= {   fd    }]
                                            { NULL[:] }
        [,CPUTIME=maxtime]
        [,classid=iocount₁ [,...,classid=iocount₃₂]]

SPOOLFILE lu&lul,pseud dev,FORM=formname [,{VFC  }]
                                            {IMAGE}

    [,{NOIMAGE}] [,{CHECKPOINT  }] [,COPIES=n] [,{HOLD   }]
      {NOVFC  }    {NOCHECKPOINT}                {RELEASE}

    [,BLOCK= blocksize/indexsize] [,{DELETE  }]  [, PRIORITY=p]
    [             1             ]   {NODELETE}
```

```
START [{ address          }] [,parameter1 ,...,parametern]
      [{ transfer address }]

[SUBMIT fd  ,DELETE] [,PRIORITY=priority]

TASK [{ taskid   }]
     [{ .BGROUND }]

               { CONTIGUOUS,fsize                                                 }
               { EC    [/[{bsize}]]    /[{isize}]]                                 }
               {          {  64 }          {  3  }                                 }
TEMPFILE lu,   { INDEX [,[{lrecl}]]  [/[{bsize}]]  [/[{isize}]]                    }
               {          { 126 }        {  1  }        {  1  }                    }
               { NB    [,[{lrecl}]]  [/[{bsize}]]  [/[{isize}]]                    }
               {          { 126 }        {  64 }        {  3  }                    }

VOLUME [voln]

WFILE [fd,]lu

               { CONTIGUOUS,fsize [,{keys}]                                        }
               {                    {0000}                                         }
               { EC    [/[{bsize}]]  [/[{isize}]]  [,[{keys}]]                     }
               {          {  64 }        {  3  }       {0000}                      }
XALLOCATE fd,  { INDEX [,[{lrecl}]]  [/[{bsize}]]  [/[{isize}]]  [,[{keys}]]       }
               {          { 126 }        {  1  }        {  1  }       {0000}       }
               { NB    [,[{lrecl}]]  [/[{bsize}]]  [/[{isize}]]  [,[{keys}]]       }
               {          { 126 }        {  64 }        {  3  }       {0000}       }
               { ITAM  [,[{lrecl}]]  [/[{bsize}]]  [[,{keys}]]                     }
               {          {  80 }        {  1  }       {0000}                      }

XDELETE fd1 [,fd2...,fdn]
```

APPENDIX B
PROGRAM DEVELOPMENT COMMAND SUMMARY

```
ADD fd [,cssprod]

          ┌ ┌   voln:    ┐ ┐ ┌ ┌    filename     ┐ ┐
COMPILE   │ {            } │ │ {                 } │
          └ └ user voln: ┘ ┘ └ └ current program ┘ ┘

           ┌ ┌   voln:    ┐ ┐ ┌ ┌    filename     ┐ ┐
COMPLINK   │ {            } │ │ {                 } │
           └ └ user voln: ┘ ┘ └ └ current program ┘ ┘

     ┌ ┌   voln:    ┐ ┐ ┌ ┌    filename     ┐ ┐
EDIT │ {            } │ │ {                 } │
     └ └ user voln: ┘ ┘ └ └ current program ┘ ┘

    ┌ ┌   voln:    ┐ ┐ ┌ ┌    filename     ┐ ┐
ENV │ { user voln: } │ │ { current program } │
    └ └   NULL     ┘ ┘ └ └                 ┘ ┘

      ┌ ┌   voln:    ┐ ┐ ┌ ┌    filename     ┐ ┐
EXEC  │ {            } │ │ {                 } │  [,"start parameters"]
      └ └ user voln: ┘ ┘ └ └ current program ┘ ┘

      ┌ ┌   voln:    ┐ ┐ ┌ ┌    filename     ┐ ┐
LINK  │ {            } │ │ {                 } │
      └ └ user voln: ┘ ┘ └ └ current program ┘ ┘

LIST

REMOVE fd

      ┌ ┌   voln:    ┐ ┐ ┌ ┌    filename     ┐ ┐
RUN   │ {            } │ │ {                 } │  [,"start parameters"]
      └ └ user voln: ┘ ┘ └ └ current program ┘ ┘
```

APPENDIX C
CSS COMMAND SUMMARY

```
 ( charlchar2₁  [charl char2₂ ...charlchar2₄] % )
%{                                              }
 ( % new delimiter                              )

        ( fd )
$BUILD  {    } [,APPEND]
        ( lu )
    .
    .
    .
$ENDB

$CLEAR

$CONTINUE

$COPY

        ( GVARIABLE    )
$DEFINE {             n},[name],operator₁  [operator₂ ...operatorₙ]
        ( IVARIABLE    )

$ELSE

$ENDC

$EXIT

$FREE varname₁  [,...,varnameₙ]

                  [(( length ))]  [          [(( length ))]]
$GLOBAL varname   [((        ))]  [,...,varname[((        ))]]
                  [((   8    ))]  [          [((   8    ))]]
```

$\underline{\$G}\text{OTO label}$

$\underline{\$L}\text{ABEL label}$

$$\text{\$IF}\left\{\begin{array}{c}\underline{\text{CH}}\text{ARACTER}\\ \underline{\text{D}}\text{ECIMAL}\\ \underline{\text{H}}\text{EXADECIMAL}\end{array}\right\}\text{arg}_1\ \underline{\text{EQ}}\text{UAL arg}_2$$

$$\text{\$IF}\left\{\begin{array}{c}\underline{\text{CH}}\text{ARACTER}\\ \underline{\text{D}}\text{ECIMAL}\\ \underline{\text{H}}\text{EXADECIMAL}\end{array}\right\}\text{arg}_1\ \underline{\text{NE}}\text{QUAL arg}_2$$

$$\text{\$IF}\left\{\begin{array}{c}\underline{\text{CH}}\text{ARACTER}\\ \underline{\text{D}}\text{ECIMAL}\\ \underline{\text{H}}\text{EXADECIMAL}\end{array}\right\}\text{arg}_1\ \underline{\text{GR}}\text{EATER arg}_2$$

$$\text{\$IF}\left\{\begin{array}{c}\underline{\text{CH}}\text{ARACTER}\\ \underline{\text{D}}\text{ECIMAL}\\ \underline{\text{H}}\text{EXADECIMAL}\end{array}\right\}\text{arg}_1\ \underline{\text{NG}}\text{REATER arg}_2$$

$$\text{\$IF}\left\{\begin{array}{c}\underline{\text{CH}}\text{ARACTER}\\ \underline{\text{D}}\text{ECIMAL}\\ \underline{\text{H}}\text{EXADECIMAL}\end{array}\right\}\text{arg}_1\ \underline{\text{LE}}\text{SS arg}_2$$

$$\text{\$IF}\left\{\begin{array}{c}\underline{\text{CH}}\text{ARACTER}\\ \underline{\text{D}}\text{ECIMAL}\\ \underline{\text{H}}\text{EXADECIMAL}\end{array}\right\}\text{arg}_1\ \underline{\text{NL}}\text{ESS arg}_2$$

$\underline{\text{\$IFE}}\text{ n}$

$\underline{\text{\$IFEX}}\text{TENSION fd}$

$IFG n

$IFL n

$IFNE n

$IFNG n

$IFNL n

$IFNULL @n

$IFNNULL @n

$IFVOLUME fd

$IFX fd

$IFNX fd

$JOB

CPUTIME=maxtime

[,classid=iocount$_1$] [,...,classid=iocount$_{32}$]

.

.

.

$TERMJOB

$\$\text{LOCAL varname}\left[\left(\begin{Bmatrix}\text{length}\\ 8\end{Bmatrix}\right)\right]\left[,\ldots,\text{varname}\left[\left(\begin{Bmatrix}\text{length}\\ 8\end{Bmatrix}\right)\right]\right]$

$NOCOPY

$PAUSE

PRIOR n

$$\underline{\text{\$}}\underline{\text{REL}}\text{EASE}\left\{\begin{matrix}\underline{\text{G}}\text{VARIABLE}\\ \underline{\text{I}}\text{VARIABLE}\end{matrix}\right\}\left[\,,\left\{\begin{matrix}n_1/n_2\\ n_1\;,\ldots,n_n\\ \text{ALL}\end{matrix}\right\}\right]$$

$SET varname=e

$$\underline{\text{S}}\text{ET }\underline{\text{C}}\text{ODE n}$$

$$\underline{\text{\$S}}\text{KIP}$$

$$\text{\$WAIT}\left[\left\{\begin{matrix}\text{n}\\ \text{1}\end{matrix}\right\}\right]$$

$$\underline{\text{\$W}}\text{RITE text }[;]$$

APPENDIX D
MTM MESSAGE SUMMARY

ACCESS PRIVILEGE ADDRESS ERROR AT XXXXXX
MEMORY FAULT ADDRESS=XXXXXX

An attempt was made to access a valid segment in an invalid mode; i.e., store into a write protected segment; execute instructions from an execute protected segment; load from a read protected segment.

ACCT-ERR

The account number specified is not a valid account.

ALIGNMENT FAULT INSTRUCTION AT XXXXXX
MEMORY FAULT ADDRESS=XXXXXX

Data instruction not properly aligned to specific fields for fullword or halfword alignment. The memory fault address is the memory location that is not properly aligned. The memory fault address is given only on Perkin-Elmer Series 3200 Machines.

ALLO-ERR TYPE=NAME

A desired filename currently exists on the specified volume.

The block size of an indexed file exceeds limit established at sysgen time.

For an indexed file, a zero logical record length or data block size was specified.

ALLO-ERR TYPE=TYPE

The volume specified is not a direct access device.

ALLO-ERR TYPE=VOL

The volume name specified, or the name it defaulted to, is not the name of any of the disks currently online.

ACCT-ERR

The account number specified is not valid.

ARGS-ERR

The amount of space between CTOP and UTOP is insufficient for placement of START command arguments by the command processor.

ARITHMETIC FAULT AT XXXXXX

A fixed or floating point error was detected at address xxxxxx, or an attempt was made to divide by zero. This only occurs on Perkin-Elmer Models 7/32 and 8/32 machines.

ASGN-ERR

The assign failed for reason denoted by TYPE field.

ASGN-ERR TYPE=BUFF

An attempt was made to assign a file when there was insufficient system space available to accommodate the FCB.

ASGN-ERR TYPE=LU

An attempt was made to assign to an lu that is greater than the maximum lu number specified at Link time.

ASGN-ERR TYPE=NAME

An assignment is being directed to a nonexistent file.

ASGN-ERR TYPE=PRIV

The privilege to assign the file or device cannot be granted. The access privileges may be incompatible with other current assignments to the same fd,

or, a request was made to assign to a disk when bare disk privileges are not enabled,

or, requested privileges may conflict with user's file access privileges (e.g., assigning system file EWO when only SRO is valid).

ASGN-ERR TYPE=PROT

The file being assigned to is unconditionally protected (read and/or write keys=X'FF') or the read/write keys specified in the ASSIGN command do not correspond to those associated with the file, and the file is conditionally protected (read and/or write keys not X'00' or X'FF').

ASGN-ERR TYPE=SIZE

An indexed file is being assigned and there is not enough room on the disk to allocate a physical block.

ASGN-ERR TYPE=SPAC

An assign is refused because the available task system space was exceeded.

ASGN-ERR TYPE=TGD

An attempt was made to assign a trap generating device.

ASGN-ERR TYPE=VOL

Volume name specified or defaulted to is not the name of any of the disks currently online.

BTCH-ERR

The batch capability was not started and is not available for a SUBMIT command.

BUFF-ERR

The expanded CSS line overflowed CSS buffer size.

CLOS-ERR

Close failed for reason denoted by TYPE field.

DELE-ERR TYPE=ACCT

An attempt was made to delete a file not on the user's private account.

DEL-ERR TYPE=ASGN

An attempt is being made to delete a file that is currently assigned, or is being processed by the CSS processor.

DELE-ERR TYPE=BUFF

There is insufficient memory available in system space to perform a delete function.

DELE-ERR TYPE=DU

An attempt was made to delete a file from a device that is not on line.

DELE-ERR TYPE=IO

An I/O error was encountered while attempting to delete a file.

DELE-ERR TYPE=NAME

File with a specified name was not found.

DELE-ERR TYPE=PROT

An attempt is being made to delete a file with nonzero protection keys.

DELE-ERR TYPE=TYPE

The volume name specified or defaulted to is not a direct access device.

| DELE-ERR TYPE=VOL

| The volume name specified or defaulted to is not the name of
| any of the disks currently online.

DUPLICATE USERNAME

Userid is already in use.

FD-ERR

The file descriptor is syntactically incorrect or invalid, or a program on the disk is being loaded without enough system space.

fd IS NOT A CONTIGUOUS FILE

The INIT command can only be used to initialize contiguous files.

FILE voln: filename. ext/acct NOT ENTERED ONTO PRINT QUEUE

A spool file was closed but the spooler task was not loaded or started.

FIXED POINT-ZERO DIVIDE ERROR AT XXXXXX
NEXT INSTRUCTION AT XXXXXXX

An attempt was made to divide by zero. Current instruction aborted, and next instruction at address xxxxxx.

FIXED POINT-OVERFLOW ERROR AT XXXXXX
NEXT INSTRUCTION AT XXXXXX

Fixed point arithmetic result is too large to be represented. Instruction aborts. Next instruction at xxxxxx.

FLOATING POINT-UNDERFLOW ERROR AT XXXXXX
NEXT INSTRUCTION AT XXXXXX

Results of floating point operation are too small to be represented. Instruction aborts. Next instruction at xxxxxx.

FLOATING POINT-OVERFLOW ERROR AT XXXXXX
NEXT INSTRUCTION AT XXXXXX

Floating point arithmetic procedure is too large to be represented. Instruction aborts. Next instruction at xxxxxx.

FLOATING POINT-ZERO DIVIDE ERROR AT XXXXXX
NEXT INSTRUCTION AT XXXXXX

An attempt was made to perform a floating point divide by zero.

FORM-ERR

The command format is invalid or invalid account number specified.

GOTO-ERR

A $LABEL that is terminating the range of the $GOTO is branching into an IF group.

ILLEGAL INSTRUCTION AT XXXXXX

The user task attempted to execute an illegal instruction at location XXXXXX.

ILLEGAL SVC-INSTRUCTION AT XXXXXX
SVC PARAMETER BLOCK AT XXXXXX

The user task attempted to execute an illegal SVC at location XXXXXX.

INVALID SEGMENT ADDRESS ERROR AT XXXXXX
MEMORY FAULT ADDRESS=XXXXXX

An attempt was made to access a memory location not within a valid mapped segment; i.e., an attempt to access a memory location outside of the task space.

INVALID ACCOUNT

Invalid or unrecognized account number.

INVALID PASSWORD

Password is invalid.

I/O-ERR

A device/file being accessed by MTM is returning a nonzero I/O status.

I/O-ERR TYPE=DU

The device is unavailable.

I/O-ERR TYPE=EOM I/O-ERR TYPE=EOF

The device reached an EOM or EOF before completing the operation.

I/O-ERR TYPE=FUNC

An invalid operation is being directed toward a device; e.g., attempting to write to a read-only device.

I/O-ERR TYPE=LU

An illegal or unassigned lu.

I/O-ERR TYPE=PRTY

A parity or other recoverable error occurred.

I/O-ERR TYPE=UNRV

An unrecoverable error occurred.

JOBS-ERR

A $JOB statement was encountered following another $JOB statement but prior to a $TERMJOB statement.

JOB NOT FOUND

The fd of job to be purged is invalid or is not in the batch job queue.

LOAD-ERR TYPE=ASGN

Load could not be accomplished because the specified fd is already exclusively assigned or could not be found.

LOAD-ERR TYPE=DU

Attempt was made to load from an unavailable device.

LOAD-ERR TYPE=I/O

An I/O error was generated during the load operation.

LOAD-ERR TYPE=LIB

The data in the loader information block is invalid. This error most frequently occurs when an attempt is made to load a task which was not built with Link.

LOAD-ERR TYPE=LOPT

Task options are incompatible with the system environment that attempts to load the task; i.e., attempt to load an e-task under MTM where e-task loading under MTM is not enabled.

LOAD-ERR TYPE=MEM

A load was attempted without enough memory specified for the task's work space.

LOAD-ERR TYPE=MTCB

The maximum number of tasks specified at sysgen time was exceeded.

LOAD-ERR TYPE=NOFP

A task requiring floating point support is being loaded, and the required floating point option is not supported in the system.

LOAD-ERR TYPE=SEG

A task requiring a task common area (TCOM) and/or a run-time library (RTL) is being loaded. The TCOM/RTL is not in the system and cannot be loaded.

LOAD-ERR TYPE=ROIO

There is an I/O error on the roll volume.

LOAD-ERR TYPE=RVOL

There is a roll file allocation or assignment error.

LU-ERR

An lu specified in an assign statement is invalid.

LVL-ERR

The number of sysgen CSS levels was exceeded.

MEMORY ERROR ON DATA FETCH AT XXXXXX
MEMORY FAULT ADDRESS=XXXXXX

Attempt was made to retrieve or to load data from a failing memory area on Perkin-Elmer Series 3200 machines. If affected memory is within task space and the operating system has memory diagnostic support, the affected page is automatically marked off, and this message is displayed:

AFFECTED MEMORY PAGE MARKED OFF AT XXXXXX

MEMORY ERROR ON INSTRUCTION FETCH AT XXXXXX
MEMORY FAULT ADDRESS=XXXXXX

A Perkin-Elmer Series 3200 machine attempted to execute an instruction from an area of memory that is failing. If affected memory is within task space and the operating system has memory diagnostic support, the affected page is automatically marked off, and this message is displayed:

AFFECTED MEMORY PAGE MARKED OFF AT XXXXXX

MEMORY PARITY ERROR AT XXXXXX

Attempt made to access nonexistent or bad memory on Models 7/32 and 8/32 machines.

MISSING PASSWORD

Password omitted.

MNEM-ERR

The command mnemonic entered is unrecognizable or a non-privileged user attempted to use a command that requires privileged status.

NOFP-ERR

No floating point support exists in the system.

NON EXISTENT SEGMENT ERROR (PST) AT XXXXXX
MEMORY FAULT ADDRESS=XXXXXX

An attempt was made to access a memory location greater than the maximum valid program address; i.e., an attempt to access a memory location outside of the task space.

NOPR-ERR

A command was entered that required more parameters than specified in the command line.

PACKED FORMAT-SIGN ERROR AT XXXXXX
MEMORY FAULT ADDRESS=XXXXXX

An illegal sign digit was detected in a packed decimal number at xxxxxx for Perkin-Elmer Series 3200 machines only.

PACKED FORMAT-DATA ERROR AT XXXXXX
MEMORY FAULT ADDRESS=XXXXXX

A data error was detected in a packed decimal number at xxxxxx for Perkin-Elmer Series 3200 machines only.

PARM-ERR

A command was entered with invalid or missing parameters.

PRIV-ERR

The access privilege mnemonic is syntactically incorrect, or an MTM user without access privileges tried to access a restricted file.

RENM-ERR TYPE=NAME

A filename already exists in the volume directory.

RENM-ERR TYPE=PRIV

The file/device cannot be assigned for ERW (required to perform the rename) because the file/device is currently assigned to at least one lu.

RENM-ERR TYPE = PROT

The protection keys of the file to be renamed are not X'0000'.

REPR-ERR TYPE=PRIV

The file/device cannot be assigned for ERW (required to carry out the reprotection) because the file/device is currently assigned to at least one lu.

ROLL-ERR

The task is currently rolled out.

SEGMENT LIMIT ADDRESS ERROR AT XXXXXX
MEMORY FAULT ADDRESS=XXXXXX

An attempt was made to access a memory location within a valid mapped segment, but the page number in the segment is greater than the largest valid page number for the segment; i.e., an attempt to access a memory location outside of the task space.

SEQ-ERR

A command was entered out of sequence or when user was not in the appropriate mode (e.g., CSS call in task loaded mode).

SIGNON REQUIRED

Attempt to enter a command before signon or a mistake in the SIGNON command.

SKIP-ERR

An attempt was made to skip beyond the end of a CSS job.

SPAC-ERR

Task exceeds established maximum system space.

SVC ADDRESS ERROR-INSTRUCTION AT XXXXXX
SVC PARAMETER BLOCK AT XXXXXX

Incorrect address of SVC parameter block at xxxxxx. The SVC parameter block must be on a fullword boundary.

SVC6-ERR TYPE=ARGS

There is insufficient room between UTOP and CTOP to contain the start option string.

SVC6-ERR TYPE=DORM

A command was issued to a specified task that is dormant.

SVC6-ERR TYPE=NMSG

| The directed task could not receive a message trap.

SVC6-ERR TYPE=PRES

| The directed task is not present in memory.

SVC6-ERR TYPE=QUE

| The message could not be queued to the directed task.

TASK-ERR

A task-related command was entered and there is no currently loaded task.

TIME-ERR

A task cannot be loaded because the user account CPU limit expired.

UNDEFINED DATA FORMAT FAULT AT XXXXXX
MEMORY FAULT ADDRESS=XXXXXX

An undefined data format/alignment fault was detected at xxxxxx for Perkin-Elmer Series 3200 machines.

USER-ERR

An invalid userid was entered in a MESSAGE command.

VOLN-ERR

The volume specified is not online or the volume name is invalid.

xxxx ERROR ON fd SECTOR n

An I/O error occurred while attempting to initialize sector n of file fd. xxxx is the type of error; it may be unrecoverable I/O, recoverable I/O, or device unavailable.

APPENDIX E
CSS MESSAGE SUMMARY

BUFF-ERR

indicates an expanded command line exceeds the CSS buffer. The task skips to $TERMJOB.

DBUF-ERR

The operators of a $DEFINE command create a result that is greater than 110 characters or the command buffer size-whichever is smaller.

DEF0-ERR

more than 8 characters specified for a keyword or a required name in the REQUIRED operator.

DEF1-ERR

an illegal character is specified in a keyword or a required name specification. A through Z are the only valid characters and they must be capital letters.

DEF2-ERR

an empty additional keyword after a quote was used in a SEARCH operator specification.

DEF3-ERR

the specified variable name is already in use.

DEF4-ERR

the REQUIRED operator must be the last operator specified in a $DEFINE command.

DEF5-ERR

divide by zero attempted.

DEF6-ERR

arithmetic fault - result is greater than Y'OFFFFFFF'.

DEF7-ERR

a negative hexadecimal value was specified. Only positive values are allowed.

FD-ERR

illegal or invalid file descriptor, or indicates not enough space to build an fd, or required file support is not in system. The task skips to $TERMJOB.

FORM-ERR

indicates a command syntax is invalid. The task skips to $TERMJOB.

GOTO-ERR

indicates a $LABEL occurred inside an IF block that was not active at the time of the $GOTO command. The task skips to $TERMJOB.

I/O-ERR

indicates an EOF was found while skipping to $ENDC, an EOF was found before a $ENDB while building a file, or a $TERMJOB was found while skipping to $ENDC within a job. The CSS skips to $TERMJOB, end of task code is set to 255, and job is ended.

JOBS-ERR

indicates a second $JOB was found before a $TERMJOB was found.

KEYW-ERR

a syntax error detected in a keyword, in a keyword parameter, or a positional parameter appears after a keyword.

LVL-ERR

indicates the CSS levels required exceed the number established at sysgen time.

MNEM-ERR

indicates the command entered is not recognized. The task skips to $TERMJOB.

NOPR-ERR

required operand for a command was not specified.

PAIR-ERR

the ' or " symbols are not matched.

PARM-ERR

indicates a command was entered with invalid or missing parameters or a variable number is not in allowed range.

REQS-ERR

REQUIRED operator is not allowed when used with new global variables in a $DEFINE command, or a syntax error was detected in a REQUIRE operator.

SEQ-ERR

indicates a command was entered out of sequence or a privileged command was used by a non-privileged user.

TASK-ERR

indicates a task-related command was entered and there is no currently loaded task. The task skips to $TERMJOB.

%REP-ERR

invalid replacement string definition or more than 4 replacement strings defined in a single character replacement command.

@SYSXXXX VARIABLE ERROR, ILLEGAL NAME

indicates that a variable was defined beginning with the reserved characters @SYS or an attempt was made to free a system variable.

@XXXX-VARIABLE ERROR, ALREADY EXISTS

indicates an attempt was made to define a local variable that already exists.

@XXXX-VARIABLE ERROR, EXCEEDS USER LIMIT

indicates that the variable limit set at sysgen was exceeded.

@XXXX-VARIABLE ERROR, DEFINITION TOO LONG

indicates that the length of the defined variable is greater than 32.

@XXXX-VARIABLE ERROR, DOES NOT EXIST

indicates an attempt to set, free, or access the value of a nonexistent variable. Also, during CSS execution, a variable definition is required.

@XXXX-VARIABLE ERROR, DEFINITION DOES NOT EXIST

indicates an attempt to set the value of a variable to the value of a second nonexistent variable.

@SYSCODE-VARIABLE ERROR, UNABLE TO ACCESS PAGE-FILE

indicates that at signon time MTM was unable to access the variable page file.

VARIABLE ERROR, VARIABLE PROCESSING NOT SUPPORTED

indicates that one of the following variable related commands was entered into a system that does not support variable processing:

- $FREE
- $GLOBAL
- $LOCAL
- $SET

VARIABLE ERROR, VARIABLE PROCESSING DISABLED

indicates that one of the following variable related commands was entered into a system with variable processing support that is disabled:

- $FREE
- $GLOBAL
- $LOCAL
- $SET

APPENDIX F
PROGRAM DEVELOPMENT MESSAGE SUMMARY

** ALTERNATE CSS REQUIRED

The fd entered with the ADD command contains a non-standard extension, and the cssprod parameter was not specified.

** COMPILE ERROR - LINK NOT EXECUTED

In a complink process, a compilation error was found, and the process aborted before the link procedure began.

** COMPILE ERRORS, LISTING ON PR:

Errors were encountered while compiling. These errors are listed on the specified pr:.

** CURRENT ENVIRONMENT - filename

The ENV command, entered without a filename, causes the name of the current environment to be displayed.

** CURRENT PROGRAM NOT SET

A filename was not specified, or no current program exists.

** EDIT - filename.ext

In the multi-module environment, the EDIT command was entered without a filename. The fd of the current source program is displayed.

** ENVIRONMENT EMPTY

The LIST command was entered, but there are no fds in the EDF.

** EXTENSION OMITTED

A filename entered with the ADD or REMOVE command did not contain the required extension.

** EXECUTION OF filename FOLLOWS:

An image program is loaded and is executing.

** FILE fd NOT FOUND

The specified filename cannot be found in the language environment.

** fd NONEXISTENT

A specified fd does not exist in the environment.

** FILENAME CONFLICT - ENTRY NOT ADDED

An attempt was made to add an already existing fd to the EDF.

** FILENAME NOT IN ENVIRONMENT

An fd specified with the REMOVE command does not exist in the EDF.

** LANGUAGE ENVIRONMENT NOT SET

A development command such as EDIT, COMPILE, COMPLINK, or EXEC was entered without first setting the language environment.

** LINK ERRORS - EXECUTION ABORTED

Program execution aborted when a link error was encountered.

** NEW ENVIRONMENT

An empty EDF has been allocated.

** NEW PROGRAM

An empty source file is allocated in the language environment.

** NO CURRENT EDF

The ENV command was entered without an EDF name, or there is no current EDF.

** NON-STANDARD EXTENSION

An attempt was made to add an fd with a non-standard language extension to the EDF without specifying a cssprod parameter.

** NOT IN MULTI-MODULE ENVIRONMENT

A command that is only meaningful in a multi-module environment was specified in a language environment.

** SOURCE FILE NOT FOUND

The specified source file cannot be found.

** SYNTAX ERROR

An fd was not specified with the ADD or REMOVE command.

** TASK fd NOT FOUND

The specified task cannot be found.

** TOO MANY ARGUMENTS

Arguments were specified in a multi-module environment.

APPENDIX G
MTM/NON-MTM TASK INTERFACE MESSAGES

G.1 $FOREGROUND TASK INTERFACE MESSAGES

xxxx-ERR SNDTID = sender task-id MSGE: received message

Where:

xxxx can be any of the following error statuses:

PARM	bad syntax in terminal-dn
TNEX	specified terminal-dn not known by MTM
TNCM	terminal not in correct mode
TASE	terminal assign error on $END message (still assigned to FOREGROUND task?)
DSTA	duplicate $STA message for the same terminal-dn received
DEND	duplicate $END message for the same terminal-dn received
MSTA	missing $STA message
MEND	missing $END message

MNEM-ERR	interface not available for normal MTM users
MOSQ-ERR	mode sequence error - terminal not in normal MTM mode
NTSK-ERR	selected task not in foreground or restricted task name
SEQ-ERR	task loaded, task executing, command substitution system (CSS), or batch mode
SMGS-ERR	send message error

#MST-ERR	missing $STA message from FOREGROUND task - terminal reassigned
#MEN-ERR	missing $END message from FOREGROUND task - terminal reassigned
TASE-ERR	FOREGROUND task assign-error
TSPC-ERR	FOREGROUND task has no more space to add the users terminal, try again later

G.2 HASP INTERFACE MESSAGE

MNEM-ERR	nonprivileged user entered the $HASPxx command
SEQ-ERR	terminal in CSS, batch mode, task loaded or executing
NTSK-ERR	no such HASPxx task-id found in foreground
USED-ERR	selected HASPxx currently being used by another MTM user
TSPC-ERR	no HASP-TUB available (more HASP tasks than specified by SGN.$HSP at MTM sysgen time)
SMGS-ERR	error on sending message to HASPxx

APPENDIX H
CONTROL SUMMARY FOR BIDIRECTIONAL INPUT/OUTPUT CONTROL (BIOC) CRT DRIVER

Bidirectional input/output control (BIOC) is a standard OS/32 terminal driver. Listed in this appendix are function control codes for the BIOC, the standard control characters generated by the use of the codes, and the functions performed. On terminals that do not generate standard control characters for any of the function keys, it is necessary to determine which key will produce the required control characters in order to invoke a desired function.

When a combination of the control key and an ASCII key cannot be accepted, BIOC will reject that combination and respond with a bell code. An example of this would be a "cancel" request (CTRL-X) on a line that has no character on it. ASCII control characters for the BIOC will not be echoed (displayed to the console) to prevent confusion between BIOC functions and terminal functions.

ASCII READ MODE:

CTRL-A (SOH) Adjust Baud Rate

The baud rate adjust function must be enabled by the system programmer before the CTRL-A can be used. When connection to a terminal is made over a dial-up line, the adjust baud rate mode is automatically entered.

To change the baud rate on a Perkin-Elmer Model 1200 terminal, for example, locate the front panel and remove the cover. It is important to know which baud rates have been made available to your terminal at system generation (sysgen) time. When this is known, depress CTRL-A and then change the baud rate setting inside the panel, using the scale depicted on the inside of the panel cover (see Figure H-1). By depressing the carriage return (CR) key repeatedly, the user will synchronize communication at the new baud rate. BIOC then responds with an asterisk (*) and continues with the mode that was in use at the time the adjust routine was begun.

6428

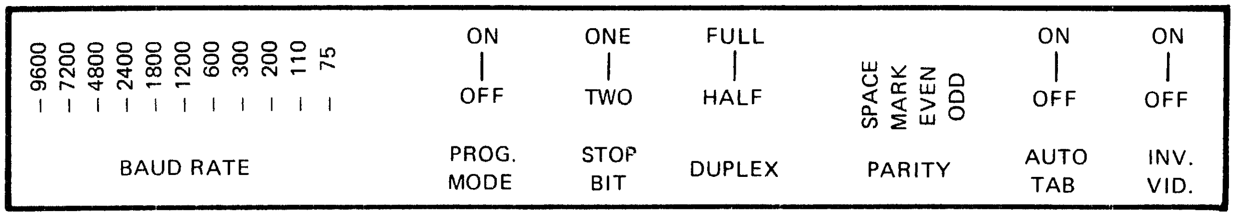


Figure H-1 Perkin-Elmer Model 1200 Mode Selectors

CTRL-B (STX) Backspace (Nondestructive)

This code causes the cursor to backspace one character for each time the code is used. To be effective, CTRL-B cannot be entered at the first character position on a line. When the cursor has been backspaced to the desired character position, the line may be changed by typing the desired characters. All other characters backspaced over can be restored and the cursor brought back to the end of the line in one of two ways:

- CTRL-F, moves the cursor forward one character at a time
- CTRL-Z, "zooms" the cursor immediately to the end of the line

CTRL-C (ETX) Capture the Last Line Entered

Entering this code will cause the last line entered (maximum of 80 characters) to be displayed on the console. By using CTRL-C repeatedly, character strings can be concatenated. If an insert or delete function is performed, the CTRL-C code will be rejected and a bell will sound to remind you that the buffer has now been overwritten. CTRL-C will also be rejected if the display of data to the console has been suppressed by the use of CTRL-E.

CTRL-D (EOT) Device Control -- Echo Only

The next character entered after the CTRL-D code will be echoed to the terminal but will not be stored in the input buffer. This function could be helpful, for example, if an auxiliary peripheral is used that requires certain control characters to be entered at the console. The CTRL-D code would prevent the peripheral control characters from being interpreted as program input.

CTRL-E (ENQ) Echo Toggle

Each entry of CTRL-E will change the current echo state from ON to OFF, or from OFF to ON. This means that data display to the console screen can be controlled. Suppression of data display is useful for entering passwords without others being able to observe them. All functions will work with echo off except CTRL-C, CTRL-R, CTRL-W, CTRL-], CTRL-^, and CTRL-_. A CTRL-M (carriage return), buffer full, or CTRL-X will turn echo back on. A CTRL-E will be rejected if the insert mode is selected.

CTRL-F (ACK) Forward Space and Restore

This code is used to restore a line that has been backspaced over by the CTRL-B, CTRL-W, or CTRL-] code. After the cursor has been moved to the desired position and the correction has been made, CTRL-F will move the cursor forward one character position at a time until it reaches the end of the line. CTRL-F will be rejected if there are no characters to be restored.

CTRL-H (BS) Backspace (Destructive)

This code is used to delete a character or characters. Unlike CTRL-B, however, any character(s) backspaced over by using the the CTRL-H code cannot be restored by using the CTRL-F or CTRL-Z codes and must be retyped. If they are not retyped, blank spaces will appear in those character positions. CTRL-H will be rejected if attempted at the first character position in a line. On most terminals the CTRL-H code can be generated by the "backspace" key.

CTRL-L (FF) Set Page Pause Line Count

To set the CRT screen display for a specific number of lines, the CTRL-L code is entered, followed by depressing the control key again with another ASCII character. The numeric value of the ASCII character will set the number of lines to be displayed. To select a count for a 24-line CRT, enter the sequence: CTRL-L, CTRL-X (X has a decimal value of 24).

The following table shows the proper combinations for line displays ranging from 1 to 24.

TABLE H-1 LINE DISPLAY COMBINATIONS

SEQUENCE	NUMBER OF LINES
CTRL-L CTRL-A	1
CTRL-L CTRL-B	2
CTRL-L CTRL-C	3
CTRL-L CTRL-D	4
CTRL-L CTRL-E	5
CTRL-L CTRL-F	6
CTRL-L CTRL-G	7
CTRL-L CTRL-H	8
CTRL-L CTRL-I	9
CTRL-L CTRL-J	10
CTRL-L CTRL-K	11
CTRL-L CTRL-L	12
CTRL-L CTRL-M	13
CTRL-L CTRL-N	14
CTRL-L CTRL-O	15
CTRL-L CTRL-P	16
CTRL-L CTRL-Q	17
CTRL-L CTRL-R	18
CTRL-L CTRL-S	19
CTRL-L CTRL-T	20
CTRL-L CTRL-U	21
CTRL-L CTRL-V	22
CTRL-L CTRL-W	23
CTRL-L CTRL-X	24

Each display of the requested number of lines is terminated with a bell sound. At this point the user may continue to the next page by entering a carriage return (CR). This will cause the same number of lines to appear; each CR will, in fact, produce that number of lines until the page pause line count is changed. To change the count, terminate write by entering ESC or Break, and enter a different sequence for the desired new line count (e.g., CTRL-L CTRL-O = 15 lines, etc.).

To cancel the page pause mode, use the sequence CTRL-L CTRL-@. If the page pause mode is not terminated within 5 minutes, BIOC will automatically continue output to prevent the terminal from being permanently tied up.

CTRL-M (CR) Terminate Read

This function is a carriage return. Entering CTRL-M indicates to BIOC that read should be terminated. If CTRL-M is entered at a location other than the end of the line, BIOC will perform a zoom to the end of the line (EOL) before storing the carriage return and terminating the read request.

CTRL-N (SO) Neutralize Selected Options Back to Default

This code is entered to reset options back to their default values. CTRL-N can be entered during read operations, during write operations, or between read and write operations. Entering CTRL-N performs the following functions:

- resets page pause to zero
- resets backspace prompt character to CTRL-H
- resets ASCII read prompt character to sysgen default
- resets backspace and CR/LF protocol to sysgen default
- resets output mode to print-on state

CTRL-O (SI) Toggle Output Between Print-on and Print-off

To suppress output in the write mode, CTRL-O is used. To resume output, this code is used again. Alternately depressing CTRL-O will cause output to terminate and resume; hence, the "toggle" characteristic. When using CTRL-O to select the print-off mode, a prompt can be immediately received by a terminate read (CTRL-M). If this is not done within 15 seconds after output ceases, BIOC will prompt and reinstate the print-on mode automatically. The print-on mode will also be reinstated upon a successful completion of a read request, or upon entering CTRL-N for a neutralize function.

CTRL-P (DLE) Set ASCII Read Prompt Character

By entering CTRL-P and any ASCII character, that character becomes the designated prompt. When making the selection, the ASCII character is not displayed to the console, but is output by BIOC upon receipt of an ASCII read request. The read prompt function can be turned off by the sequence CTRL-P CTRL-X. To reset the ASCII read prompt character to the sysgen default, enter CTRL-N.

CTRL-Q (DC1) Removed from Input to Allow X-ON/X-OFF Flow Control

CTRL-R (DC 2) Reprint Entered Line

When this code is entered, the current cursor location within the line will determine the number of characters that will be reprinted on the next line. All characters, including blank spaces, to the left of the cursor will be reprinted. The CTRL-R function will be rejected if the echo state is turned off (see CTRL-E).

The CTRL-R function is especially useful for hardcopy terminals where corrections are made over the existing typed lines. To view a "clean" line after all corrections have been made, CTRL-P is used.

CTRL-S (DC 3) Removed from Input to Allow X-ON/X-OFF Flow Control

CTRL-T (DC4) Single Character Transparent Mode

The use of this code will allow the entry of function control characters into the input buffer. The next character entered after a CTRL-T will be entered directly into the input buffer.

CTRL-W (ETB) Word Backspace (Nondestructive)

CTRL-W causes the cursor to be backspaced (nondestructively) to the nearest nonalphabetic character. Thus, CTRL-W allows the cursor to backspace over one complete word, rather than one character, as with CTRL-B. Words backspaced over may be restored by the use of CTRL-F or CTRL-Z. CTRL-W will be rejected if attempted at the beginning of a line.

CTRL-X (CAN) Cancel Current Input Line

All characters previously entered on the current line will be deleted upon use of the code. Characters may not be restored with the CTRL-F or CTRL-Z functions. If no characters are on the line, CTRL-X will be rejected. CTRL-X will turn echo back on if it has been turned off with CTRL-E.

CTRL-Z (SUB) "Zoom" to Furthest End of Line

CTRL-Z can be used to restore a line that has been backspaced over by CTRL-B, CTRL-W, or CTRL-]. CTRL-Z will cause the cursor to "zoom" to the end of the line, but will be rejected if there are no characters to be restored.

CTRL-] (GS) Backward Character Search (Nondestructive)

This code serves to locate a specific character on the current line. For example, to find the character $, enter CTRL-]$. BIOC will backspace until the first $ is found. To find any additional dollar signs on the same line, the code must be entered again for each time the $ symbol appears. Characters backspaced over may be restored by using CTRL-F or CTRL-Z. CTRL-] will be rejected if attempted at the beginning of the line.

CTRL-^ (RS) Toggle Between Insert-on and Insert-off

Each CTRL-^ toggles from insert-on to insert-off or from insert-off to insert-on. When the insert mode is selected, characters typed will be inserted in front of the character currently over the cursor. The insert mode may be selected only when the cursor is positioned at a location other than the end of the line and the echo state is on. The insert mode will be terminated by another CTRL-^ or by any command that takes the cursor position to the end of the line (e.g., CTRL-Z). The CTRL-C and CTRL-E functions are not valid while in the insert mode. All other functions are valid if the cursor is not in motion. All data entered while the cursor is in motion will be ignored until the cursor has stopped.

CTRL-_ (US) Delete Character

Each CTRL-_ deletes the character currently over the cursor. The delete code is valid only when the cursor is positioned at a location other than the end of the line and the echo state is on. Characters entered while the cursor is in motion will be ignored.

WRITE MODE:

BREAK

This key terminates write with the Break status.

ESC

This key terminates write with the Break status.

CTRL-Q

This code resumes write after write has been suspended by CTRL-T or CTRL-S functions.

CTRL-R

This code resumes write after write has been suspended by CTRL-T or CTRL-S functions.

CTRL-S

This code suspends write until write is resumed by CTRL-R or CTRL-Q or until the BREAK or ESC key is depressed.

CTRL-T

This code suspends write until write is resumed by CTRL-R or CTRL-Q or until the BREAK or ESC key is depressed.

INDEX

/@INPUT control statement 7-2
/@SUBMIT control statement 7-3

A

Access privileges
  compatibility 2-6
Access protection of disks 1-8
ACCOUNT operator 6-24
ADD command 4-12
ALLOCATE command 2-2
ASSIGN command 2-6
Assigning logical units 4-10
Authorization 1-3

B

Batch commands
  INQUIRE 5-3
  LOG 5-5
  PURGE 5-7
  SIGNOFF 5-8
  SIGNON 5-9
  SUBMIT 5-11
Batch job error handling 5-13
Batch job priority
  establishing 5-12
Batch job submission
  using the spooler 5-13
Batch jobs 7-3
Batch processing
  affect of restricted
   disks on 5-13
  batch task pause option 5-13
  commands 5-1
  error handling 5-13
  introduction 5-1
Batch task pause option 5-13
BFILE command 2-12
BIAS command 2-13
Bidirectional input/output
 control H-1
  control summary for H-1
BIOC. See bidirectional
 input/output control.
BREAK command 2-14
Break key
  using the 1-10
BRECORD command 2-15
BUILD command 2-16
$BUILD command 6-18

C

CAL 4-1
Calling a CSS file 6-2
  use of parameters 6-3
CANCEL command 2-18
Character replacement command 6-14
Checkpointing 2-56
  5-6
CLEAR operator 6-36
CLOC 2-37
CLOSE command 2-19
COBOL 4-2
Command conventions 1-11
Command mode 1-6
Command substitution system
  ACCOUNT operator 6-24
  call parameters 6-3
  calling a CSS file 6-2
  character replacement
   command 6-14
  CLEAR operator 6-36
  command summary C-1
  commands 6-13
  computation operator 6-30
  conversion operator 6-30
  CURRENT operator 6-37
  DCOMPUTE operator 6-30
  DHCONVERT operator 6-32
  DVOLUMENAME operator 6-38
  end of task code testing
   commands 6-61
  EXTENSION operator 6-25
  file descriptor operators 6-24
  file existence testing
   commands 6-62
  FILENAME operator 6-26
  general description 6-1
  HCOMPUTE operator 6-33
  HDCONVERT operator 6-34
  line expansion 6-12
  logical IF commands 6-60
  LOGICAL operators 6-29
  message summary E-1
  mode 1-6
  other operators 6-35
  parameter existence
   testing 6-62
  PRIOR command 6-52
  REQUIRED operator 6-39
  SEARCH operator 6-40
  SET CODE command 6-56
  signon 1-7
  STRING operator 6-43
  use of keywords 6-5
  VOLUMENAME operator 6-28
  $BUILD command 6-18
  $CLEAR command 6-20
  $CONTINUE command 6-21
  $COPY command 6-22
  $DEFINE command 6-23
  $ELSE command 6-63
  $ENDB command 6-18
  $EXIT command 6-45
  $FREE command 6-46
  $GLOBAL command 6-47
  $GOTO command 6-64
  $IF...EQUAL command 6-69

Command substitution system (Continued)
  $IF...GREATER command 6-69
  $IF...LESS command 6-70
  $IF...NEQUAL command 6-69
  $IF...NGREATER command 6-69
  $IF...NLESS command 6-70
  $IFEXTENSION command 6-66
  $IFVOLUME command 6-67
  $JOB command 6-48
  $LABEL command 6-64
  $LOCAL command 6-50
  $NOCOPY command 6-22
  $PAUSE command 6-51
  $RELEASE command 6-53
  $SET command 6-55
  $SKIP command 6-57
  $TERMJOB command 6-48
  $WAIT command 6-58
  $WRITE command 6-59
Commands
  introduction 2-1
COMPILE command 4-14
COMPIlE command
  language format 4-14
COMPILE command
  multi-module format 4-14
COMPLINK command
  language format 4-18
  multi-module format 4-18
Computation operator 6-30
  DCOMPUTE 6-30
  DHCONVERT 6-32
  HCOMPUTE 6-33
  HDCONVERT 6-34
CONTINUE command 2-20
Conversion operator 6-30
  DCOMPUTE 6-30
  DHCONVERT 6-32
  HCOMPUTE 6-33
  HDCONVERT 6-34
Creating a data file 4-4
Creating a source program 4-1
CRT driver. See also BIOC.
CSS. See command substitution system.

CTOP 2-38
CTSW 2-37
CURRENT operator 6-37
$CLEAR command 6-20
$CONTINUE command 6-21
  6-58
$COPY command 6-22

D

DCOMPUTE Operator 6-30
Default lu assignments 4-10
Default variable settings 4-10
DELETE command 2-21
DHCONVERT operator 6-32
DISPLAY ACCOUNTING command 2-24
DISPLAY command 2-22
DISPLAY DEVICES command 2-25
  sample display 2-25
DISPLAY DFLOAT command 2-27
DISPLAY FILES command 2-28
DISPLAY FLOAT command 2-34
DISPLAY LU command 2-35
DISPLAY PARAMETERS command
  fields 2-37
  task option bit definitions 2-38
DISPLAY REGISTERS command 2-42
DISPLAY TIME command 2-43
DISPLAY USERS command 2-44
DVOLUMENAME operator 6-38
$DEFINE command 6-10
  6-11
  6-23
  ACCOUNT operator 6-24
  CLEAR operator 6-36
  computation operator 6-30
  conversion operator 6-30
  CURRENT operator 6-37
  DCOMPUTE operator 6-30
  DHCONVERT operator 6-32
  DVOLUMENAME operator 6-38
  EXTENSION operator 6-25
  file descriptor operators 6-24
  FILENAME operator 6-26
  HCOMPUTE operator 6-33
  HDCONVERT operator 6-34
  LOGICAL operators 6-29
  other operators 6-35
  REQUIRED operator 6-39
  SEARCH operator 6-40
  STRING operator 6-43
  VOLUMENAME operator 6-28

E

ECM. See environmental control monitor.
EDF. See environment descriptor file.
EDIT command 4-21
ENABLE command 2-45
End of task code testing 6-60
End of task code testing commands 6-61
  $IFE n 6-61
  $IFG n 6-61
  $IFL n 6-61
  $IFNE n 6-61
  $IFNG n 6-61
  $IFNL n 6-61
ENDB command 2-16
ENV command 4-6
  4-23
Environment descriptor file 4-6
ENVIRONMENT= parameter 5-10
Environmental control monitor 3-5
EXAMINE command 2-46
EXEC 4-4
  4-5
EXEC command 4-25

Executing a program 4-4
Executing multiple programs
 as a single program 4-6
EXTENSION operator 6-25
$ELSE 6-60
$ELSE command 6-63
$ENDB command 6-20
$ENDC 6-60
$EXIT command 6-45

F

fds. See file descriptors.
FFILE command 2-48
File conventions 1-11
File descriptor operators 6-24
 EXTENSION 6-25
 FILENAME 6-26
 VOLUMENAME 6-28
File descriptors 1-12
 format 1-13
 parameters 1-13
File existence testing 6-60
File existence testing
 commands 6-62
 $IFNX 6-62
 $IFX 6-62
File initialization. See
 INIT command.
Filename operator 6-26
Foreground task
 interfacing with 3-1
 programming details 3-2
 restricted task-ids 3-1
Foreground task mode 1-6
FORT 4-1
FORTO 4-1
FORTZ 4-1
FRECORD command 2-49
$FOREGROUND task interface
 messages G-1
$FREE command 6-46

G

Global variables 6-9
Group account numbers 1-11
$GLOBAL command 6-10
 6-47
$GOTO command 6-64

H

Hasp interface
 linking the HASP task 3-4
Hasp interface messages G-2
Hasp interface mode 1-6
HCOMPUTE operator 6-33
HDCONVERT operator 6-34
HELP command 2-50
Help facility 1-7

I

INIT command 2-52
Input spooling 7-2
 /@INPUT control statement 7-2
 /@SUBMIT control
 statement 7-3
 control card statements 7-2
INQUIRE command 5-3
Interactive task to terminal
 mode 1-6
ITC/RELIANCE interface 3-5
$IF...EQUAL command 6-69
$IF...GREATER command 6-69
$IF...LESS command 6-70
$IF...NEQUAL command 6-69
$IF...NGREATER command 6-69
$IF...NLESS command 6-70
$IFE n 6-61
$IFEXTENSION command 6-66
$IFG n 6-61
$IFL n 6-61
$IFNE n 6-61
$IFNG n 6-61
$IFNL n 6-61
$IFNNULL 6-62
$IFNULL 6-62
$IFNX 6-62
$IFVOLUME command 6-67
$IFX 6-62

J

$JOB command 6-48

K

Keywords
 defining 6-5
 examples of illegal 6-7
 referencing within the
 CSS 6-7
 rules for use of 6-6
 valid examples of 6-6

L

Language tab character 4-2
LINK command 4-28
 link sequences 4-29
LIST command 4-32
LOAD command 2-53
Loading a task 1-7
Loading tasks. See TASK
 command.
Local variables 6-9
LOG command 2-55
 5-5
 checkpointing 2-56
Logical IF commands 6-60
 comparing two arguments 6-68
 end of task code testing 6-60

Logical IF commands
(Continued)
file existence testing 6-60
parameter existence testing 6-60
$ELSE 6-60
$ENDC 6-60
$IF...EQUAL 6-69
$IF...GREATER 6-69
$IF...LESS 6-70
$IF...NEQUAL 6-69
$IF...NGREATER 6-69
$IF...NLESS command 6-70
LOGICAL operators 6-29
$LABEL command 6-64
$LOCAL command 6-10
6-50

M

MACRO 4-1
MESSAGE command 2-57
MODIFY command 2-58
Modifying a program 4-5
MPRI 2-38
MTM. See multi-terminal monitor.
MTM/non-MTM task interface messages G-1
Multi-terminal monitor
ALLOCATE command 2-2
ASSIGN command 2-6
authorization 1-3
BFILE command 2-12
BIAS command 2-13
BREAK command 2-14
BRECORD command 2-15
BUILD command 2-16
CANCEL command 2-18
CLOSE command 2-19
command summary A-1
CONTINUE command 2-20
DELETE command 2-21
devices 1-3
DISPLAY ACCOUNTING command 2-24
DISPLAY command 2-22
DISPLAY DEVICES command 2-25
DISPLAY DFLOAT command 2-27
DISPLAY FILES command 2-28
DISPLAY FLOAT command 2-34
DISPLAY LU command 2-35
DISPLAY PARAMETERS command 2-37
DISPLAY REGISTERS command 2-42
DISPLAY TIME command 2-43
DISPLAY USERS command 2-44
dynamic functions 1-1
ENABLE command 2-45
ENDB command 2-16
environments 1-4
EXAMINE command 2-46
FFILE command 2-48
FRECORD command 2-49
HELP command 2-50

Multi-terminal monitor
(Continued)
INIT command 2-52
introduction 1-1
LOAD command 2-53
LOG command 2-55
MESSAGE command 2-57
message summary D-1
MODIFY command 2-58
operation 1-1
OPTIONS command 2-60
PASSWORD command 2-61
PAUSE command 2-62
PREVENT command 2-63
PRINT command 2-64
priority 6-52
privileged users 1-3
PUNCH command 2-66
RENAME command 2-70
REPROTECT command 2-71
REWIND command 2-72
RVOLUME command 2-73
RW command 2-72
SEND command 2-76
SET GROUP command 2-77
SET KEYOPERATOR command 2-79
SET LOG command 2-56
SET PRIVATE command 2-81
SIGNOFF command 2-83
SIGNON command 2-84
SPOOLFILE command 2-86
START command 2-89
tailoring functions 1-1
TASK command 2-90
TEMPFILE command 2-91
transmitting messages 1-4
user commands 2-1
user information 1-2
VOLUME command 2-94
WFILE command 2-96
XALLOCATE command 2-97
XDELETE command 2-100
$RELEASE command 2-68
Multi-terminal monitor environments
batch 1-5
interactive 1-4
MUSP 2-38
MXSP 2-38

N

Naming global variables 6-10
Naming local variables 6-10
Naming new global variables 6-10
Naming new internal variables 6-10
New global variables 6-9
6-10
6-11
referencing the value of 6-11
New internal variables 6-9
6-10
6-11
referencing the value of 6-11
NLU 2-38
$NOCOPY command 6-22

O

OPTIONS command 2-60
OS/32 spooler 7-1
Other operators 6-35
  CLEAR 6-36
  CURRENT 6-37
  DVOLUMENAME 6-38
  REQUIRED 6-39
  SEARCH 6-40
  STRING 6-43
Output spooling 7-5
  assigning logical units 7-5

P,Q

Parameter existence testing 6-60
Parameter existence testing commands 6-62
  $IFNNULL 6-62
  $IFNULL 6-62
PASCAL 4-2
PASSWORD command 2-61
PAUSE command 2-62
PREVENT command 2-63
PRINT command 2-64
PRIOR command 6-52
Private account numbers 1-11
Privileged users 1-3
Program development
  assigning logical units 4-10
  command availability 4-8
  command summary B-1
  commands 1-8
  creating a data file 4-4
  creating a source program 4-1
  default lu assignments 4-10
  default variable settings 4-10
  error recovery 4-10
  executing a program 4-4
  executing multiple programs 4-6
  language commands 4-1
  language tab character 4-2
  message summary F-1
  modifying a program 4-5
  re-executing a modified prgoram 4-5
  sample sessions 4-36
  source file language extensions 4-2
Program development commands 4-11
  ADD 4-12
  COMPILE 4-14
  COMPLINK 4-18
  EDIT 4-21
  ENV 4-23
  EXEC 4-25
  LINK 4-28
  LIST 4-32
  REMOVE 4-33
  RUN 4-34
Prompt conventions 1-9
PUNCH command 2-66
PURGE command 5-7
$PAUSE command 6-51

R

Re-executing a modified program 4-5
REMOVE command 4-33
RENAME command 2-70
REPROTECT command 2-71
REQUIRED operator 6-39
Reserved variables 6-13
Restricted task-ids 3-1
REWIND command 2-72
RPG 4-2
RUN command 4-34
RVOLUME command 2-73
RW command 2-72
$RELEASE command 2-68
  6-10
  6-53

S

Sample program development sessions 4-36
SCTASK 2-53
SEARCH operator 6-40
Security 1-8
SEND command 2-76
SET CODE command 6-56
SET GROUP command 2-77
SET KEYOPERATOR command 2-79
  6-6
SET LOG command 2-56
SET PRIVATE command 2-81
SIGNOFF command 2-83
  5-8
SIGNON command 2-84
  5-9
SLOC 2-38
Source file language extensions 4-2
SPL/32 and MTM interaction 7-8
SPL/32 spooler 2-86
  7-1
  7-7
  MTM interaction 7-8
SPOOLFILE command
  SPL/32 spooler 2-86
Spooling 1-8
  /@INPUT control statement 7-2
  /@SUBMIT control statement 7-3
  control card statements 7-2
  errors 7-6
  input 7-2
  OS/32 spooler 7-1
  output 7-5
  SPL/32 and MTM interaction 7-8
  SPL/32 spooler 7-1
  7-7
START command 2-89
STAT 2-37
STRING operator 6-43
SUBMIT command
  batch job priority 5-11
SVOL 2-38
System account numbers 1-12

$SET command 6-55
$SKIP command 6-57

T

TASK command 2-90
Task executing mode 1-6
Task interfaces
   foreground tasks 3-1
   HASP tasks 3-1
   ITC/RELIANCE tasks 3-1
   messages G-1
   non-MTM 3-1
Task loaded mode 1-6
TEMPFILE command 2-91
Terminal conventions 1-10
Terminal modes 1-6
   command mode 1-6
   CSS mode 1-6
   foreground task mode 1-6
   hasp interface mode 1-6
   interactive task to terminal 1-6
   task executing mode 1-6
   task loaded mode 1-6
Terminal users
   number of 1-4
TOPT 2-37
Transmitting messages 1-4
$TERMJOB command 6-48

U

UBOT 2-38
USERINIT.CSS 2-84
USSP 2-37
UTOP 2-38

V

Variables
   global 6-9
   local 6-9
   naming global 6-10
   naming local 6-10
   naming new global 6-10
   naming new internal 6-10
   new global 6-9
   new internal 6-9
   reserved 6-13
   types of 6-9
   use of 6-9
VOLUME command 2-94
VOLUMENAME operator 6-28

W

Wait status bit definitions 2-40
WFILE command 2-96
$WAIT command 6-58
$WRITE command 6-59

X,Y,Z

XALLOCATE command 2-97
XDELETE command 2-100

CUT ALONG LINE

PUBLICATION COMMENT FORM

Please use this postage-paid form to make any comments, suggestions, criticisms, etc. concerning this publication.

From ______________________ Date ______________________

Title ______________________ Publication Title ______________________

Company ______________________ Publication Number ______________________

Address ______________________

____

____

FOLD FOLD

Check the appropriate item.

☐	Error	Page No. ________	Drawing No. ________
☐	Addition	Page No. ________	Drawing No. ________
☐	Other	Page No. ________	Drawing No. ________

Explanation:

FOLD FOLD

Fold and Staple
No postage necessary if mailed in U.S.A.

6434

STAPLE

ACC = 10
GRP = 0
PAS = SYSOP
ID = WALT

STAPLE

FOLD

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 22 OCEANPORT, N.J.

POSTAGE WILL BE PAID BY ADDRESSEE

PERKIN-ELMER

Computer Systems Division
2 Crescent Place
Oceanport, NJ 07757

TECH PUBLICATIONS DEPT. MS 322A

FOLD

FOLD

STAPLE

STAPLE

643

PERKIN-ELMER
Technical Systems Division

D O C U M E N T A T I O N   C H A N G E   N O T I C E

The purpose of this documentation change notice (DCN) is to provide a quick and efficient way of making technical changes to manuals before they are formally updated or revised.

The manual affected by these changes is:

48-043 F00 R01 OS/32 Multi-Terminal Monitor (MTM) Reference Manual

---

- Page 2-87

  In the Parameters column, add IMAGE directly under VFC. Then change the last sentence of the VFC paragraph to:

  If IMAGE is specified, there is no VFC for the device assigned to the specified lu.

- Page 2-87

  In the Parameters column, add NOIMAGE directly above NOVFC. Then change the paragraph to:

  turns the VFC option or IMAGE option off for the assigned lu. NOVFC is the default option.

- Page 2-87

  Add the following parameters and descriptions before COPIES=:

CHECKPOINT	turns on checkpointing for the assigned lu. This is the default option. The global checkpoint option must be on.
NOCHECKPOINT	turns off checkpointing for the assigned lu.